TAMPEREEN TEKNILLINEN YLIOPISTO

Heikki Hautala

Androidin fragmentaatio – ongelmia ja ratkaisuja

Diplomityö

Tarkastaja: professori Tommi Mikkonen
Tarkastaja ja aihe hyväksytty Tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 5.6.2013

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
HAUTALA, HEIKKI: Androidin fragmentaatio – ongelmia ja ratkaisuja
Diplomityö, 45 sivua
Kesäkuu 2015
Pääaine: Ohjelmistotuotanto
Tarkastaja: Tommi Mikkonen
Avainsanat: Android, fragmentaatio

Android on Googlen kehittämä avoimeen lähdekoodiin perustuva käyttöjärjestelmä, joka on suunnattu kosketusnäytöllisille mobiililaitteille. Android-laitteiden valmistajia on paljon, ja valmistajilla saattaa olla kymmeniä eri laitemalleja. Vuonna 2013 eri laitemalleja oli lähes 12000. Syy tähän on se, että Androidin lähdekoodi on avoin. Eri valmistajien ja laitteiden valtava määrä on johtanut siihen, että sovelluskehittäjien on entistä vaikeampi taata sovellustensa toimivuus jokaisessa olemassa olevassa laitteessa. Laitteiden näyttökoot vaihtelevat suuresti, mutta sovellusten tulisi kuitenkin näyttää hyvältä kaiken kokoisilla näytöillä. Laitevalmistajat joutuvat myös toteuttamaan itse oheislaitteidensa ajureita, jolloin oheislaitteet saattavat käyttäytyä eri tavalla eri laitteissa. Lisäksi eri Android-versioita on paljon käytössä yhtä aikaa. Uusimmissa laitteissa on yleensä jokin Androidin uusimmista versioista, mutta edelleen myydään edullisia laitteita, joissa on jopa vuosia vanha versio eikä virallista päivitystä ole saatavilla. Yllä mainittujen seikojen vuoksi Android on **fragmentoitunut**. Android-fragmentaatio tarkoittaa siis sitä, että yhtä aikaa käytössä on paljon kooltaan, laitteistoltaan ja käyttöjärjestelmäversioltaan eroavia Android-laitteita.

Tässä diplomityössä selvitetään Android-fragmentaatiosta johtuvia ongelmia, jotka tulisi ottaa huomioon Android-sovelluskehitysessä, sekä pyritään tuomaan esille erilaisia ratkaisuja, joita soveltaen kehitettävat Android-sovellukset toimisivat yhtä hyvin ja näyttäisivät yhtä hyviltä erilaisilla Android-laitteilla.

Työssä käy ilmi, että Google tarjoaa monipuoliset Android-kehitystyökalut, joita oikein käyttämällä on mahdollista torjua suuri osa Android-fragmentaation tuomista ongelmista. Se vaatii kuitenkin paljon työtä toteutus- ja testausvaiheessa. Osa ongelmista sen sijaan saattaa vaatia hyvinkin laitteesta tai laitevalmistajasta riippuvia räätälöityjä toteutuksia. Kaikkia mahdollisia ongelmia on mahdotonta luetella, koska jokainen uusi Android-laite ja -versio voi tuoda uusia ongelmia.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
HAUTALA, HEIKKI: Android fragmentation – problems and solutions
Master of Science Thesis, 45 pages
June 2015
Major: Software Engineering
Examiner: Tommi Mikkonen
Keywords: Android, fragmentation

Android is Google's open source operating system for mobile devices with a touchscreen. There are many Android device manufacturers, and each one may have dozens of different device models available. In 2013 there were almost 12000 different device models in use. The reason for this is the fact that Android is an open source operating system. Android software developers have a hard time making their applications to work properly on every Android device, because of the large number of different manufacturers and devices. Nevertheless, Android applications should look good on every available screen size. Device manufacturers have to implement their own peripheral drivers, which means that the peripheral equipment may operate differently on different devices. In addition, there are many different versions of Android in use. The most recent devices are usually running one of the most recent version of Android. However, there are still cheap devices on sale which are running few years old versions of Android, and will not get an official update. Because of the above-mentioned facts, Android is **fragmented**. Android fragmentation means that simultaneously there are many Android devices with different screen size, equipment and OS version.

This master's thesis will explain the problems caused by Android fragmentation that should be taken into consideration in Android software development. It will also present various solutions that will help developers to develop Android applications that look good and work properly on different devices.

This thesis will discover that a large part of the problems of fragmentation can be solved by the correct use of the development tools provided by Google. However, it will require a lot of work in the implementation and testing phase. Some of the problems may require very specific manufacturer or device related implemenetations. It is impossible to list all the problems caused by Android fragmentation, because every new Android device and version can possibly introduce a completely new set of problems.

ALKUSANAT

Ennen tämän työn kirjoittamista olin tehnyt useita Android-sovelluksia töissä ja vapaa-ajalla. Yritin keksiä vuoden ajan hyvää diplomityöaihetta, kunnes kävin professori Tommi Mikkosen juttusilla. Kun kerroin kokemuksestani ja kiinnostuksestani Androidiin, niin Tommi keksi heti ehdottaa Androidin fragmentaatiota aiheeksi. Aloitin työn kirjoittamisen vappuna 2013. Kului kaksi vuotta, ja nyt vappuna 2015 työ on valmis.

Kiitän professori Tommi Mikkosta mielenkiintoisesta aiheesta ja rakentavasta palautteesta, Juha Riippiä työn ohjauksesta sekä kotijoukkoja kannustuksesta.

"Hiljaa hyvä tulee ja hitaasti kiirutta piretään"
- Suomalainen sananparsi

Tampereella 01.05.2015

Heikki Hautala

SISÄLLYS

TERMIT JA NIIDEN MÄÄRITELMÄT

Android NDK	*Android Native Development Kit* Androidin natiivi kehitystyökalusarja.
API	*Application Programming Interface*, ohjelmointirajapinta.
ART	*Android Runtime*, Dalvik-virtuaalikoneen korvannut ajoaikaympäristö Android 5.0:ssa.
CDD	*Compatibility Definition document*, Androidin yhteensopivuusohjelman määrittelydokumentti.
CTS	*Compatibility Test Suite*, Testisarja Android-yhteensopivuuden testaamiseksi.
CDMA	*Code Division Multiple Access*, koodijakokanavointi.
EXIF	*Exchangeable image file format*, standardi kuvatiedostojen metatiedolle.
MIME	*Multipurpose Internet Mail Extensions*, sisällön tiedostomuodon kertova internet standardi.
OHA	*Open Handset Alliance*, 84 yrityksen yhteenliittymä, joka kehittää avoimia standardeja mobiililaitteille.
Resoluutio	Fyysisten pikselien määrä näytöllä.
SMS	*Short Message Service*, matkapuhelinten tekstiviestijärjestelmä.
USB	*Universal Serial Bus*, sarjaväyläarkkitehtuuri.
WAP	*Wireless Application Protocol*, langattomien sovellusten protokolla.

1. JOHDANTO

Android on Googlen kehittämä avoimeen lähdekoodiin perustuva käyttöjärjestelmä, joka on suunnattu kosketusnäytöllisille mobiililaitteille. Androidin ensimmänen versio julkaistiin syyskuussa vuonna 2008. Sen jälkeen Android on kehittynyt nopeasti. Tämän työn kirjoitushetkellä uusin versio on 5.0, joka julkaistiin lokakuussa vuonna 2014.

Android-laitteiden valmistajia on paljon ja valmistajilla saattaa olla kymmeniä eri laitemalleja. Vuonna 2013 eri mallisia laitteita oli lähes 12000. Syy tähän on se, että Androidin lähdekoodi on avoin. Eri valmistajien ja laitteiden valtava määrä on johtanut siihen, että sovelluskehittäjien on entistä vaikeampi taata sovellustensa toimivuus jokaisessa olemassa olevassa laitteessa. Laitteiden näyttökoot vaihtelevat suuresti, mutta sovellusten tulisi kuitenkin näyttää hyvältä kaiken kokoisilla näytöillä. Laitevalmistajat joutuvat myös toteuttamaan itse oheislaitteidensa ajureita, jolloin oheislaitteet saattavat käyttäytyä eri tavalla eri laitteissa. Lisäksi eri Android-versioita on paljon käytössä yhtä aikaa. Uusimmissa laitteissa on yleensä Androidin viimeisin versio, mutta edelleen myydään edullisia laitteita, joissa on useitakin vuosia vanha Android-versio eikä virallista päivitystä ole saatavilla. Yllä mainittujen seikkojen vuoksi Android on **fragmentoitunut**. Android-fragmentaatio tarkoittaa siis sitä, että yhtä aikaa käytössä on paljon kooltaan, laitteistoltaan ja käyttöjärjestelmäversioltaan eroavia Android-laitteita.

Tässä diplomityössä selvitetään Android-fragmentaatiosta johtuvia ongelmia, jotka tulisi ottaa huomioon Android-sovelluskehityksessä, sekä pyritään tuomaan esille erilaisia ratkaisuja, joita soveltaen kehitettävät Android-sovelluksen toimisivat yhtä hyvin ja näyttäisivät yhtä hyviltä erilaisilla Android-laitteilla.

Työssä käy ilmi, että Google tarjoaa monipuoliset Android-kehitystyökalut, joita oikein käyttämällä on mahdollista torjua suuri osa Android-fragmentaation tuomista ongelmista. Se vaatii kuitenkin paljon työtä toteutus- ja testausvaiheessa. Osa ongelmista sen sijaan saattaa vaatia hyvinkin laitteesta tai laitevalmistajasta riippuvia räätälöityjä toteutuksia. Kaikkia mahdollisia ongelmia on mahdotonta luetella, koska jokainen uusi Android-laite ja -versio voi tuoda uusia ongelmia.

Luku 2 esittelee Android-käyttöjärjestelmän. Luvussa käydään läpi Androidin historia, versiot ja ominaisuudet, Google Play -sovelluskauppa, Androidin arkkitehtuuri, ohjelmointiympäristöt ja -kielet, sekä sovelluskomponentit. Luvun lopuksi selvitetään Android-projektin ja -manifestin rakenne, sekä selitetään käyttöliittymän luonnin perusteet. **Luvussa 3** selitetään fragmentaatio yleisesti, miten fragmentaatio liittyy Androdiin ja mitä hyviä puolia fragmentaatiosta voi olla. Lopuksi esitellään Googlen Android-yhteensopivuusohjelma. **Luvussa 4** käydään läpi eri fragmentaatiotasoihin liittyviä ongelmia, sekä niihin liittyviä ratkaisuja. Lopuksi esitellään kaksi testausta tukevaa työkalua ja opastetaan hallitsemaan tuettujen laitteiden listaa Google Play:ssa. **Luvussa 5** luodaan työn yhteenveto.

2. JOHDATUS ANDROIDIIN

Android on avoimen lähdekoodin Linuxiin perustuva [1] käyttöjärjestelmä. Se on suunniteltu ensisijaisesti mobiililaitteille, joissa on kosketusnäyttö.

Tässä luvussa käydään läpi Androidin perusteet, jotta seuraavien lukujen sisältö olisi helpommin ymmärrettävissä. Luku alkaa Androidin historiasta ja etenee teknisempiin yksityiskohtiin loppua kohti.

2.1 Androidin historia

Vuoden 2003 lokakuussa Californian Palo Altossa perustettiin start-up yritys nimeltä Android Inc. Yrityksen perustajina toimivat Andy Rubin, Rich Milner, Nick Sears ja Chris White [2]. Yrityksen päämääränä oli Rubinin mukaan kehittää *"älykkäämpiä mobiililaitteita, jotka tiedostavat paremmin käyttäjiensä sijainnin ja mieltymykset."* [3]

Heinäkuussa 2005 Google osti start-up yrityksiä. Yksi niistä oli *Android Inc*. Kesällä vuonna 2007 heti *Applen* iPhonen julkaisun jälkeen huhut Googlen omasta mobiililaitteesta voimistuivat. Huhujen mukaan uuden laitteen nimi olisi *gPhone*. Saman vuoden marraskuussa kuitenkin selvisi, etteivät Googlen suunnitelmat olleet koskeneet vain uutta laitetta, vaan myös uutta avoimen lähdekoodin käyttöjärjestelmää, joka kilpailisi Symbianin ja muiden samankaltaisten käyttöjärjestelmien kanssa. Androidia ei ollut kehittänyt ainostaan Google, vaan sen johtama yhteenliittymä nimeltä Open Handset Alliance (OHA), johon kuuluivat Googlen lisäksi mm. Samsung, LG, HTC, T-Mobile ja muita alan yrityksiä. Google oli ollut suurin Androidin kehittäjä.[3]

Vuoden 2007 marraskuussa, viikko Androidin julkistuksen jälkeen OHA julkaisi Android-kehittäjille kehitystyökalut. Seuraavan vuoden helmikuussa Qualcomm, Texas Instruments ja monet muut laitevalmistajat saivat ensimmäiset Androidia tukevat piirisarjansa valmiiksi.[3]

Ensimmäinen Android-laite oli HTC:n valmistama HTC Dream. Sen huhuttiin tulevan myyntiin vasta vuoden 2009 puolella. Tieto viivästymisestä sai osakekurssit

horjumaan, ja pian HTC:n rahoitusjohtaja julkaisi tiedotteen, jonka mukaan kuluttajat saisivat ensimmäiset laitteet jo vuoden 2008 viimeisellä neljänneksellä. Laite tuli myyntiin syyskuussa. Yhdysvalloissa se sai nimeksi T-Mobile G1.[3]

2.2 Androidin versiot ja ominaisuudet

Android on julkaisunsa jälkeen kehittynyt nopeasti. Uusia päivityksiä on tullut muutaman kuukauden välein lisäten Androidiin uusia ominaisuuksia ja korjaten sen virheitä. Android 1.5 oli ensimmäinen päivitys, jolle Google antoi lempinimen. Nimeksi tuli *Cupcake*. Siitä lähtien jokainen korkeamman tason päivitys on saanut oman lempinimensä. Versiot ovat kaikki nimetty eri jälkiruokien mukaan. Androidin versiot ja niiden mukanaan tuomat tärkeimmät ominaisuudet on esitetty taulukossa 2.1.

Uusien päivityksien myötä Android on saanut uusia ominaisuuksia ja rajapintoja. Muutamaa poikkeusta lukuunottamatta mikään päivitys ei ole poistanut olemassaolevaa ominaisuutta tai rajapintaa, mikä tärkoittaa sitä, että varhaisemmilla kehitystyökaluilla tehdyt sovellukset toimivat myös uudemmilla Android versioilla. [4]

Androidin lähdekoodit on julkaistu pääosin *Apache 2.0*-lisenssillä. On myös joitakin poikkeuksia, kuten Linux-ytimen paikkaukset, jotka on julkaistu *GPLv2*-lisenssillä. *Apache 2.0*-lisenssin ehdot sallivat lähdekoodin käytön avoimen sekä suljetun koodin kehittämiseen. Lisenssi kuitenkin vaatii alkuperäisen tekijänoikeuden huomautuksen säilyttämisen. [5]

2.3 Google Play

Google Play on mediakauppa, joka aloitti toimintansa vuoden 2008 lokakuussa. Silloin sen nimi oli vielä *Android Market*, ja se tarjosi vain ilmaisia Android sovelluksia. Maksujärjestemä otettiin ensimmäisen kerran käyttöön Yhdysvalloissa ja Englannissa helmikuussa 2009 [7]. Nimi Google Play otettiin käyttöön vuoden 2012 maaliskuussa. Uuden nimen tarkoitus oli poistaa mielikuva, jonka mukaan se tarjoaisi sisältöa vain Android-laitteille [8].

Google Play tarjoaa ilmaisia ja maksullisia sovelluksia, kirjoja, lehtiä, elokuvia, TV-sarjoja, musiikkia sekä laitteita. Sisältötyyppien saatavuus on maakohtaista. Suomessa Google Play tarjoaa nykyään jo kaikkia sisältötyyppejä.

Android-käyttöjärjestelmän mukana tulee *Play Store* -sovellus, jolla sisältöä pääsee lataamaan suoraan laitteelta. Musiikille, videoille ja kirjoille on myös omat

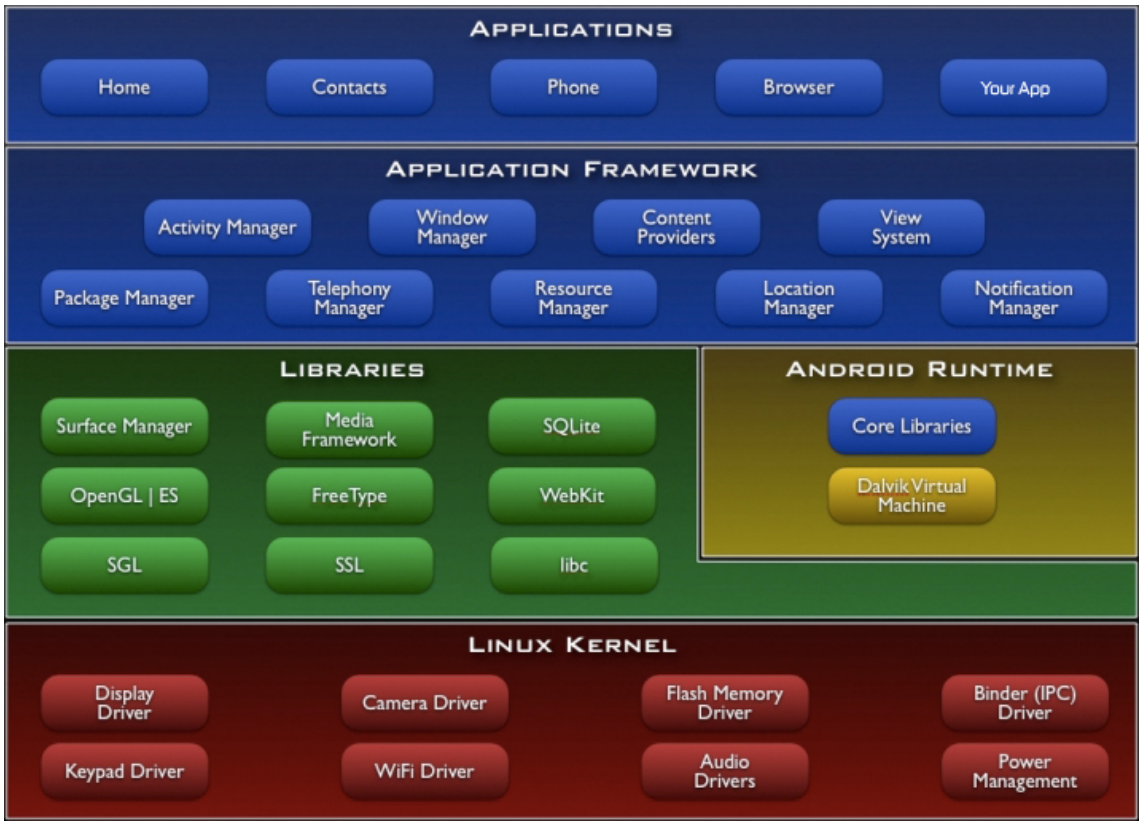
Taulukko 2.1: Androidin versiot [6].

Versio	**API**	**Julkaistu**	**Tärkeimmät uudistukset**
1.0	1	Syyskuu 2008	Ensimmäinen versio
1.1	2	Helmikuu 2009	SMS- ja WAP-viestien lähetys, MMS-viestien liitteiden tallennus.
1.5 Cupcake	3	Huhtikuu 2009	Widgetit, virtuaalinen näppäimistö ja tuki kolmannen osapuolen toteutuksille, videoiden tallennus ja toisto, Bluetooth, selaimen kopio&liitä, SIM työkalut.
1.6 Donut	4	Syyskuu 2009	Pikahaku, Text-to-speech, eleet, tuki uusille näyttökoille, tuki CDMA puhelimille.
2.0 Eclair	5	Lokakuu 2009	Tuki useille käyttäjätileille, tuki sisäänrakennettulle salamavalolle, HTML5 tuki, Bluetooth 2.1.
2.0.1	6	Joulukuu 2009	-
2.1	7	Tammikuu 2010	Liikkuvat taustakuvat.
2.2 Froyo	8	Toukokuu 2010	Wifi hotspot -tuki, usean kielen tuki virtuaalinäppäimistölle, sovellusten asennus ulkoiselle muistille, tiedon varmuuskopiointi, virheraportointi.
2.3 Gingerbread	9	Joulukuu 2010	VoIP-tuki, NFC-tuki, tuki gyroskoopille ym. uusille sensoreille, tuki usealle kameralle, tuki erittäin suurille näytöille.
2.3.3	10	Helmikuu 2011	Laajempi NFC tuki.
3.0 Honeycomb	11	Helmikuu 2011	Fragmentit, Action Bar, leikepöytä, Drag&Drop.
3.1	12	Toukokuu 2011	USB API, tuki ulkoisille USB laitteille, widgettien koon muutaminen, sovellusten pysäytystila.
3.2	13	Heinäkuu 2011	Screen Support API - mm. näyttökokokohtaiset resurssit.
4.0 Ice cream sandwich	14	Lokakuu 2011	Oma profiili, kasvontunnistus, Wifi-Direct, tuki lämpötila- ja kosteusantureille, räätälöitävä käynnistin.
4.0.3	15	Joulukuu 2011	Social stream API.
4.1 Jellybean	16	Heinäkuu 2012	-
4.2	17	Marraskuu 2012	Daydream, Photo Sphere panoraamakuvaus, widgettien lisääminen näyttölukkoon, käyttäjäprofiilit, sisäkkäiset fragmentit.
4.3	18	Heinäkuu 2013	Käyttäjäprofiilien rajoittaminen, OpenGL ES 3.0 tuki, Bluetooth Smart tuki, Photo Sphere.
4.4 Kitkat	19	Lokakuu 2013	Tulostusrajapinta, askelmittarituki, koko näytön tila, Chromium WebView, ruudun nauhoitus, infrapunalaitetuki.
5.0 Lollipop	21	Lokakuu 2014	Material design, ART, OpenGL ES 3.1 tuki.

katselu- ja kuuntelusovelluksensa: *Play Music*, *Play Movies* sekä *Play Books*. *Play Store*-sovelluksen lisäksi sisältö voi ladata laitteelle myös etänä web-selaimella. Tämä vaatii laitteen liittämisen *gmail*-tunnukseen.

Kehittäjät voivat jakaa kehittämiään sovelluksia Google Play:n kautta. Kehittäjätilin luonti maksaa 25 Yhdysvaltain dollaria. Toisin kuin Applen kehitystilissä, maksu ei ole vuosittainen, vaan se maksetaan vain kerran. Google ottaa maksullisten sovellusten tuotoista 30% käsittelymaksun, jolloin kehittäjälle jää 70% [9]. Google Play:sta ostetun sovelluksen voi palauttaa 2 tunnin sisällä, jolloin ostoksesta saa rahat takaisin. Saman sovelluksen voi palauttaa vain kerran [10].

2.4 Androidin arkkitehtuuri



Kuva 2.1: Android ohjelmistopinon arkkitehtuuri [1].

Kuvan 2.1 komponenttikaavio esittää Androidin ohjelmistoarkkitehtuurin. Se koostuu Linux ytimestä, C:llä ja C++:lla kirjoitetuista natiiveista Android-kirjastoista, Android Runtime:sta sekä Java:lla kirjoitetuista sovelluskehyksestä ja itse Android-sovelluksista.

Android perustuu Linux 2.6 -ytimeen, johon Google on tehnyt joitain muutoksia. Ydin sisältää kaikki tärkeät laitteistoajurit ja mahdollistaa siten sovellusten ja

laitteiston välisen kommunikaation. Ydin vastaa mm. muistinhallinnasta, prosessienhallinnasta, verkkoyteyksistä ja turvallisussasetuksista. [1]

Androidin natiivikirjastot mahdollistavat erityyppisten tietojen käsittelyn. Esimerkiksi *Media Framework* sisältää useita mediakoodekkeja, jotka mahdollistavat eri mediaformaattien tallennuksen ja toiston. *SQLite* on Androidin käyttämä tietokantojenhallintajärjestelmä. [1]

Android Runtime sisältää Dalvik-virtuaalikoneen sekä Javalla kirjoitetut ydinkirjastot. Se on optimoitu matalan suorituskyvyn laitteille, joissa on rajoitetusti muistia. Se mahdollistaa myös useamman virtuaalikoneen yhtäaikaisen ajon, mikä edistää turvallisuutta, eristämistä, muistinhallintaa ja jolla on luotu tuki säikeistykselle [1]. Android Runtime:n ydinkirjastot sisältävät osajoukon Java SE:n (Java Standard Edition) kirjastoista. Pois on jätetty kirjastoja, joita Androidissa ei tarvita (esim. javax.print) tai joihin on olemassa korvaavia parempia kirjastoja [11].

Android-sovelluskehyksen osat näkyvät parhaiten sovelluskehittäjille. Tärkeimpiä osia ovat sovelluksen näyttöjä hallinnoiva *Activity Manager*, tiedon jakamisen sovellusten välillä mahdollistavat *Content Providerit*, puheluita hallinnoiva *Telephony Manager*, GPS-sijaintien käsittelyyn käytettävä *Location Manager*, sekä *Resource Manager*, jonka avulla sovellus pääsee käsiksi sen omiin resursseihin kuten kuva- ja äänitiedostoihin. [1]

2.5 Ohjelmointiympäristöt

Ennen joulukuuta 2014 Googlen suosittelema ohjelmointiympäristö eli IDE Android-kehitykseen oli *Eclipse*, joka on *Eclipse Foundationon* avoimen lähdekoodin tuote. Eclipsen konfigurointi Android-kehitystä varten vaati Android SDK:n sekä ADT (*Android Developer Tools*) Plugin:in asentamisen Eclipseen. Myöhemmin Androidin kehittäjäsivustolta oli ladattavissa Android SDK Bundle, johon oli esiasennettu kaikki tarvittavat osat.

Joulukuussa 2014 Google julkisti *Android Studio* -IDE:n version 1.0, joka on siitä lähtien ollut Googlen suosittelema IDE Android-kehitykseen [12]. Android Studio perustuu *Jetbrains*-yhtiön kehittämään *IntelliJ IDEA*:an, jonka *Community*-versio on ilmainen ja sisältää myös tuen Android-kehitykselle [13].

Android Studio ja IntelliJ IDEA tarjoavat Eclipseä enemmän ominaisuuksia. Ne tukevat kirjastoriippuvuuksia ja sovellusvarianttien luontia sekä tarjoavat kehittyneemmän kooditäydennyksen ja koodin uudelleenjärjestelyn kuin Eclipse.

2.6 Ohjelmointikielet

Koska Android perustuu Linuxiin, sen natiivi ohjelmointikieli on C/C++, mutta koska Android SDK on toteutettu Javalla, sen ohjelmointikielenä käytetään Javaa. Android SDK mahdollistaa myös C/C++-ohjelmoinnin sen natiivin ohjemointirajapinnan NDK:n avulla. Suurin osa sovelluksista ei hyödy NDK:n käytöstä, koska se vain lisää lähdekoodin monimutkaisuutta eikä yleensä tuo tehokkuushyötyjä. NDK:ta kannattaa käyttää silloin kun sovellukseen halutaan liittää valmiita C:llä tai C++:lla kirjoitettuja komponentteja, tai jos sovelluksen täytyy suorittaa nopeasti prosessori-intensiivisiä laskutoimituksia kuluttamatta paljoa muistia. Jos sovelluksen täytyy nopealla tahdilla luoda ja poistaa olioita, on se parempi tehdä natiivikoodilla, koska Javassa oliot luodaan aina dynaamisesti ja niiden tuhoaminen muistista on Javan roskienkeruun vastuulla. Kun muisti täytyy nopeasti, alkaa roskienkeruu keskeytellä sovelluksen suoritusta poistaakseen turhia olioita. Tämä näkyisi esimerkiksi grafiikkaintensiivisessä pelissä ruudunpäivitysen tökkimisenä.

2.7 Androidin sovelluskomponentit

Androidin sovelluskomponentteja on neljä. Niitä ovat **aktiviteetit** (engl. *activity*), **palvelut** (engl. *service*), **sisällöntarjoajat** (engl. *content provider*) ja **kuuntelijat** (engl. *broadcast receiver*). Aktiviteetit, palvelut ja kuuntelijat aktivoidaan asynkronisilla viesteillä, joita kutsutaan *aikeiksi* (engl. *intent*).[15]

2.7.1 Aikeet

Aie on tietorakenne, joka sisältää kuvauksen suoritettavasta operaatiosta. *Aikeita* käytetään viestinvälityksessä aktiviteettien, palveluiden ja kuuntelijoiden välillä. *Aikeita* voidaan ajatella vietinvälittäjinä, jotka pyytävät muita komponentteja tekemään jonkin operaation.

Sovellus voi tiedottaa mielenkiintoisista tapahtumista muita sovelluksia tai sovelluksen komponentteja lähettämällä *aikeita*. *Aikeeseen* kytketään operaation nimi, jolloin ne kuuntelijat, jotka on rekisteröity kuuntelemaan samaa operaatiota, voivat vastaanottaa *aikeen* ja toimia sen mukaisesti. *Aie* lähetetään käyttämällä funktiota *Context.sendBradcast()*:

```
// Aikeen lähetys kuuntelijoille
Intent intent = new Intent();
intent.setAction("com.example.myapp.mybroadcast");
context.sendBroadcast(intent);
```

2.7.2 Aktiviteetit

Aktiviteettia voisi kutsua myös ruuduksi, sillä se vastaa sovelluksessa yhtä ruudun täyttävää näkymää. Vastaava luokka iOS-alustalla olisi *ViewController* ja Windows Phone -alustalla *Page*. Jos sovelluksella on näkyvä käyttöliittymä, sisältää se ainakin yhden aktiviteetin. Sovelluksen aktiviteetit pidetään pinossa ja jokainen sovelluksessa käynnistetty aktiviteetti asetetaan pinon päällimmäiseksi. Aktiviteetti suljetaan yleensä painamalla fyysistä tai ohjelmallista *takaisin* painiketta, jolloin pinon päällimmäinen eli näkyvä aktiviteetti tuhotaan ja pinossa seuraavaksi olevaan aktiviteettiin palataan. Aktiviteetteja voidaan kuitenkin hallita hyvin monipuolisesti jos tarve vaatii. Jokainen aktiviteetti periytetään luokasta *android.app.Activity* tai sen aliluokasta. Aktiviteetti käynnistetään kutsumalla funktiota *Context.startActivity()*. Sille annetaan parametrina *aie*, joka sisältää tiedon käynnistettävästä aktiviteetista: [15]

```
// Aktiviteetin käynnistäminen
Intent intent = new Intent(context, MyActivity.class);
context.startActivity(intent);
```

2.7.3 Palvelut

Palvelu on taustalla omassa säikeessään ajettava ohjelma, jolla ei ole näkyvää käyttöliittymää. Sitä käytetään aikaa vievien operaatioiden suorittamiseen, jotta sovelluksen näkyvä käyttöliittymä pysyisi reagoivana. Palvelu voidaan käynnistää ja jättää ajoon, jolloin se jatkaa ajoa, vaikka sen käynnistänyt sovellus sammutettaisiin. Toinen vaihtoehto on sitoa palvelun elinkaari sen käynnistävään sovellukseen, jolloin palvelu sammuu kun sovellus sammuu. Palvelua voitaisiin käyttää esimerkiksi musiikin soittamiseen tai tiedostojen lataamiseen internetistä. Jokainen palvelu periytetään luokasta *android.app.Service* tai sen aliluokasta. Palvelu käynnistetään käyttäen *Context*-luokan funktiota *startService()* : [15]

```
// Palvelun käynnistäminen
Intent intent = new Intent(context, MyService.class);
context.startService(intent);
```

2.7.4 Sisällöntarjoajat

Sisällöntarjoajia käytetään tiedon jakamiseen sovellusten välillä. Sisällöntarjoaja voi tarjota tietoa suoraan tiedostoista, SQLite-tietokannasta tai mistä tahansa lähteestä, johon sovellus pääsee käsiksi. Android käyttää sisällöntarjoajaa esimerkiksi sen puhelinluettelosovelluksessa, joka tarjoaa muille sovelluksille rajapinnan puhelinluettelon tietoihin. Rajapinnan kautta voidaan lukea, muokata ja poistaa tietoa, jos sisällöntarjoaja niin sallii. Jokainen sisällöntarjoaja periytetään luokasta *android.content.ContentProvider* tai sen aliluokasta.[15]

2.7.5 Kuuntelijat

Kuuntelija on komponentti, joka pystyy vastaanottamaan viestejä järjestelmältä ja muilta sovelluksilta. Sovellus voi sisältää useita kuuntelijoita ja kukin kuuntelija voidaan asettaa kuuntelemaan kaikkia tai vain tiettyjä viestejä. Kuuntelijan avulla sovellus voi esimerkiksi reagoida internetyhteyden tilan muuttumiseen ja näyttää käyttäjälle ilmoituksen yhteyden katkeamisesta. Sovellus voi myös itse lähettää viestejä muille sovelluksille sen omista tapahtumista. Kuuntelijat periytetään luokasta *android.content.BroadcastReceiver* tai sen aliluokasta. Listauksessa 2.1 on esimerkki kuuntelijasta, jolla voidaan tunnistaa Internet-yhteyden katkeaminen. [15]

Listaus 2.1: Internet-yhteyden katkeamisen tunnistaminen kuuntelijalla.

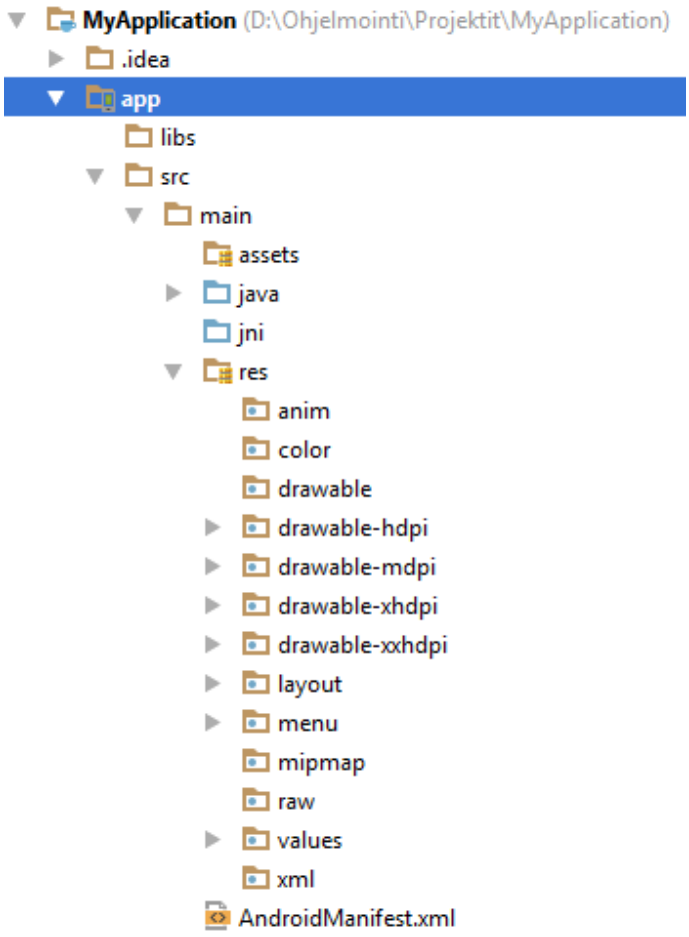
```
// ConnectivityReceiver.java:
public class ConnectivityReveicer extends BroadcastReceiver {

   @Override
   public void onReceive(Context context, Intent intent) {

      // Internet-yhteyden tila on muuttunut.
      ConnectivityManager manager =
         (ConnectivityManager) context.getSystemService(
            Context.CONNECTIVITY_SERVICE);
      NetworkInfo info = manager.getActiveNetworkInfo();
      if (info == null || !info.isConnectedOrConnecting()) {

         // Ilmoitetaan yhteyden katkeamisesta.
         Toast.makeText(context, "Internet connection lost!",
                        Toast.LENGTH_SHORT).show();
      }
   }
}

...

// Kuuntelijan ja sen aie-suodattimien asetus AndroidManifest.xml:ssä:
<application>
   ...
   <receiver android:name="ConnectivityReceiver">
      <intent-filter>
         <action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
         <action android:name="android.net.wifi.WIFI_STATE_CHANGED"/>
      </intent-filter>
   </receiver>
</application>
```

2.8 Android-projekti

Android-projekti pitää sisällään sovelluksen lähdekoodin ja käytetyt resurssitiedostot, kuten kuvat, äänet ja aktiviteettien käyttöliittymäelementtien sommitelmat (engl. *layout*). Kuvassa 2.2 on esitetty Android-projektin sovellus-moduulin tiedostohierarkia Android Studio -ympäristössä.

Taulukko 2.2: Android-projektin applikaatio-moduulin kansiorakenne ja tiedostot Android Studio -ympäristössä [16].

Kansio	**Tiedostot**
libs	Yksityiset kirjastot, joita sovellus käyttää.
src	Projektin lähdekoodit.
main/assets	Raa'at resurssitiedostot, jotka tallennetaan lopulliseen *.apk*-pakettiin sellaisenaan säilyttäen tiedostonimet. Koodissa näihin tiedostoihin pääsee käsiksi *AssetManager*-luokan avulla.
main/java	Java Lähdekoodit.
main/jni	Lähdekoodit, jotka käyttävät JNI:tä (Java Native Interface).
main/res	Resurssitiedostot kuten kuvat, käyttöliittymäelementtien ladontaohjeet, käyttöliittymätekstit ja -värit.
res/anim	XML-tiedostot, jotka käännetään animaatio-olioiksi.
res/color	XML-tiedostot, jotka määrittävät käytettäviä värejä.
res/drawable	Bittikarttatiedostot sekä XML-tiedostot, jotka käännetään piirrettäviksi olioiksi.
res/mipmap	Sovelluksen käynnistysikonit.
res/layout	XML-tiedostot, jotka määrittävät käyttöliittymäkomponenttien ladontaohjeet aktiviteeteille tai niiden osille.
res/menu	XML-tiedostot, jotka määrittävät sovelluksen valikoiden sisällöt.
res/raw	Mielivaltaiset raa'at resurssitiedostot (esim. äänitiedostot), joihin pääsee koodissa käsiksi generoidun *R*-luokan kautta.
res/value	XML-tiedostot, jotka käännetään monenlaisiksi resursseiksi. Näihin tiedostoihin ei voi viitata tiedostonimellä, vaan tietyn XML-elementin nimellä. Elementtejä ovat mm. merkkijonot (<*string*>), värit (<*color*>) ja ulottuvuudet (<*dimen*>).
res/xml	Sekalaiset XML-tiedostot, joilla säädetään sovelluskomponentteja.



Kuva 2.2: Android-projektin applikaatio-moduulin tiedostohierarkia

2.9 Android-manifesti

Android-manifesti on *AndroidManifest.xml* -niminen XML-tiedosto projektin juurihakemistossa. Manifesti sisältää sovelluksen olennaisimmat tiedot, joita Android-järjestelmä tarvitsee pystyäkseen suorittamaan sovellusta. Manifestiin määritellään seuraavat asiat [17]:

- Sovelluksen Java-paketin nimi. Nimi on ainutlaatuinen tunniste sovellukselle.
- Sovelluksen versionumero ja versionimi. Versionumero on sovelluksen sisäinen versio, jota ei näytetä käyttäjille. Versionimi on mielivaltainen merkkijono, joka näytetään käyttäjälle mm. Google Play -kaupassa ja laitteen sovellusten hallinnassa.
- Kaikki käytetyt sovelluskomponentit. Jokainen aktiviteetti, palvelu, kuuntelija ja sisällöntarjoaja tulee määritellä erikseen.

- Sovelluksen tarvitsemat oikeudet, kuten Internetin käyttö tai paikannustietojen luku.
- Varhaisin ohjelmointirajapinta eli API, jota sovellus tukee.
- Laiteominaisuudet, joita sovellus käyttää. Näitä ovat esimerkiksi kamera ja Bluetooth.
- Linkitettävät kirjastot (poislukien Android-kirjasto, johon jokainen android-projekti on linkitetty automaattisesti).

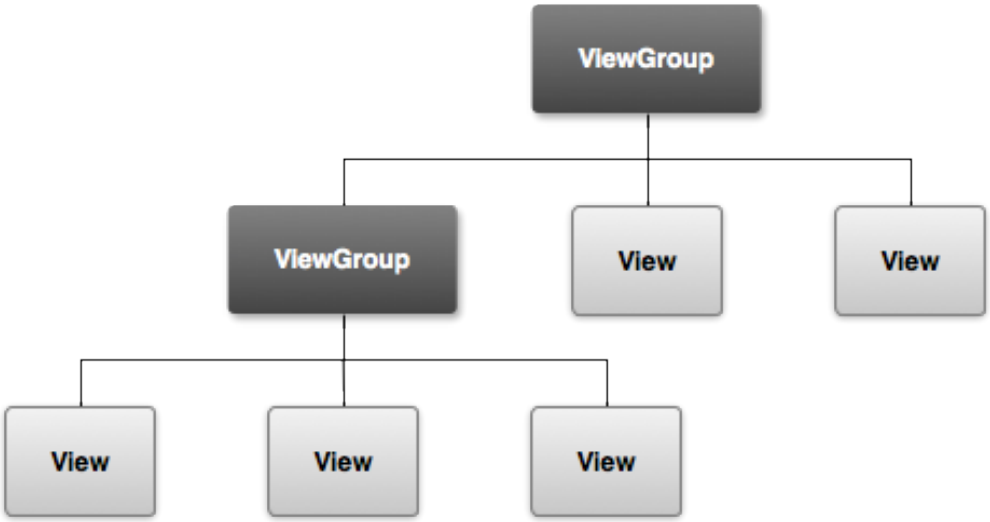
Listauksessa 2.2 on esimerkki yksinkertaisesta manifestista. Tämän manifestin omaavan sovelluksen voi asentaa vain laitteille, jossa on Android 2.1 (API 7) tai uudempi. Sovellus tarvitsee Internet- ja Bluetooth-oikeudet. Sovelluksen pystyisi asentamaan myös laitteelle, jossa ei ole Bluetoothia, koska ko. ominaisuutta ei ole merkitty pakolliseksi. Sovellus sisältää kaksi aktiviteettia, joista *FirstActivity* on merkitty ensin käynnistettäväksi kun sovellus käynnistetään.

Listaus 2.2: AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp"
    android:versionCode="1" android:versionName="1.0">

    <uses-sdk android:minSdkVersion="7" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-feature
        android:name="android.hardware.bluetooth" android:required="false"/>

    <application
        android:label="@string/app_name" android:icon="@drawable/app_icon">
        <activity
            android:name=".FirstActivity" android:label="@string/app_name">
            <intent-filter>
                <action
                    android:name="android.intent.action.MAIN" />
                <category
                    android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".SecondActivity"/>
    </application>
</manifest>
```

2.10 Käyttöliittymän luonti

Käyttöliittymät Androidissa koostuvat näkymien (*View*) ja näkymäryhmien (*ViewGroup*) hierarkkisesta puusta. Tätä puuta on havainnollistettu kuvassa 2.3. Näkymä on olio, joka piirtää jotain näytölle. Näkymäryhmä on olio, joka pitää sisällään näkymiä ja muita näkymäryhmiä. Näkymäryhmän tehtävä on järjestellä sisältönsä. Yleisin tällainen näkymäryhmä on *LinearLayout*. Se järjestää lapsinäkymänsä vierekkäin tai allekkain riippuen konfiguraatiosta. Toinen usein käytetty näkymäryhmä on *RelativeLayout*, jolla voi nimensä mukaisesti järjestellä lapsinäkymät suhteessa toisiinsa. [18]



Kuva 2.3: Käyttöliittymäsommitelman hierarkia [18].

Käyttöliittymähierakian voi rakentaa Java-koodissa näkymä kerrallaan, mutta helpompi ja nopeampi tapa on luoda XML-tiedosto. Käyttöliittymän määrittäminen XML-tiedostoissa myös parantaa koodin luettavuutta, koska silloin se ei sisällä näkymien luonti- ja asettelukoodia. Lisäksi XML-tiedostoja on mahdollista uudelleen käyttää muissa näkymissä. Käyttöliittymäkomponentit voi pääsääntöisesti myös konfiguroida kokonaan XML-attribuutteja käyttäen. XML-tiedoston elementit ovat näkymiä ja näkymäryhmiä, joiden attribuutit määräävät niiden ominaisuudet [18]. Listauksessa 2.3 on esimerkki yksinkertainen esimerkki XML-tiedostosta, joka kuvaa käyttöliittymän, jossa on allekkain teksti ja painike.

Listaus 2.3: Yksinkertainen näkymä XML-tiedostona.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/confirm_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Press, OK to confirm." />

    <Button
        android:id="@+id/ok_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="OK" />
</LinearLayout>
```

XML-ladontatiedostoista voi ajoaikana muodostaa näkymäolioita käyttäen siihen tarkoitettua luokkaa nimeltä *LayoutInflater*. Aktiviteetin (2.7.2) näkymää asetettaessa käytetään kuitenkin sen omaa funktiota *setContentView()*, jolle annetaan suoraan viittaus XML-tiedostoon.

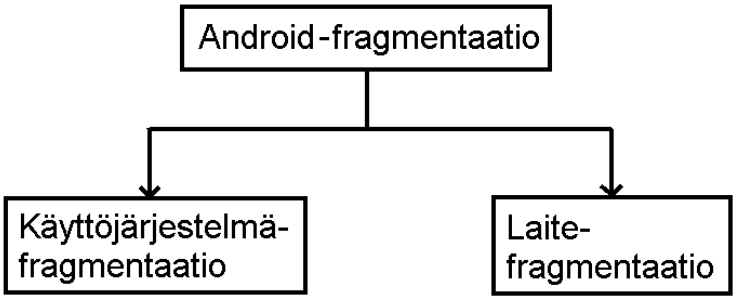
3. ANDROIDIN FRAGMENTAATIO

Tässä luvussa selitetään fragmentaatiota yleisesti ja käydään läpi Android-fragmentaation eri tasot.

3.1 Fragmentaation tasot

Sana **fragmentaatio** tarkoittaa pirstaloitumista, hajoamista pieniin osiin. Esimerkiksi tietokoneen muistin fragmentoituminen tarkoittaa sitä, että kun tiedosto tallennetaan muistiin, niin se joudutaan tallentamaan useampaan eri paikkaan muistissa, jos sille ei löydy yhtä yhtenäistä tarpeeksi suurta paikkaa. Kun näin käy ajan saatossa yhä useammalla tiedostolle, niin lopputuloksena muisti on täynnä pieniä tiedoston osia. Kun jokin tiedosto halutaan lukea muistista, sen osat täytyy etsiä useasta paikasta, mikä vie enemmän aikaa verrattuna siihen, että tiedosto haettaisiin kokonaan yhdestä paikasta.

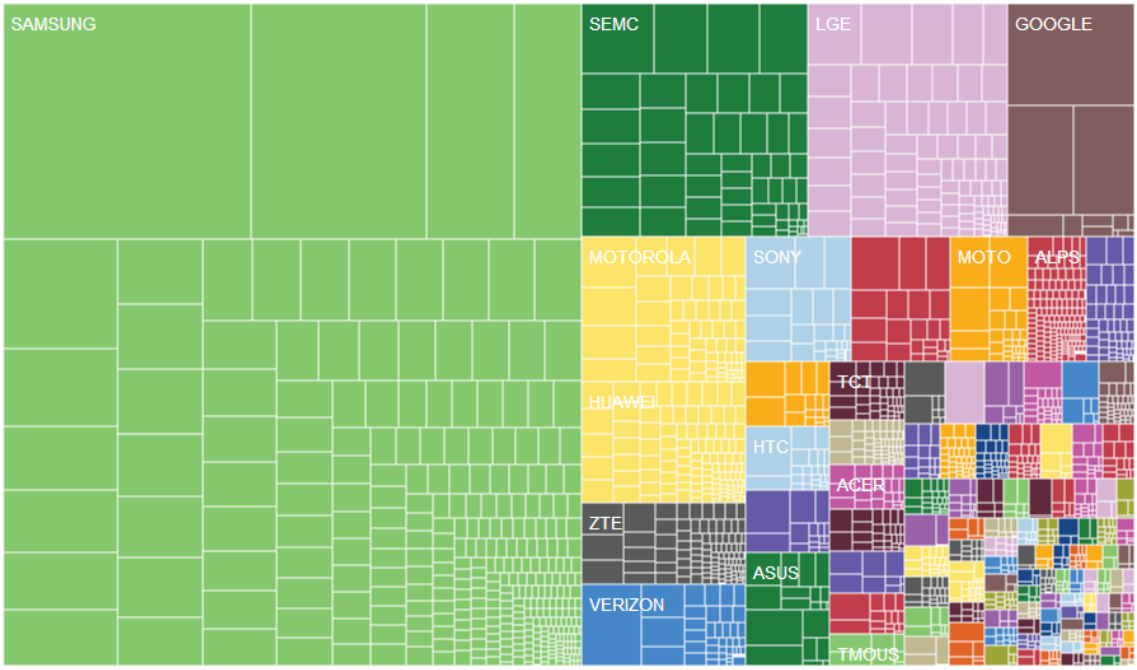
Androidin fragmentaatio voidaan jakaa kuvan 3.1 mukaisesti kahteen osaan, käyttöjärjestelmä- ja laite-fragmentaatioon.



Kuva 3.1: Androidin fragmentaation tasot [19]).

Androidin fragmentoituminen johtuu siitä, että Android on avoimen lähdekoodin käyttöjärjestelmä, josta kuka vain voi muokata oman versionsa ja alkaa valmistamaan Android-laitteita. Tästä johtuen Android-laitteiden valmistajia on paljon, kuten kuvasta 3.2 näkee. Suuri laitekirjo johtaa siihen, että laitteiden Android

versiot sekä ominaisuudet kuten näyttökoko ja oheislaitteiden määrä vaihtelevat suuresti. Android-sovellusten kehittäminen niin, että ne toimivat ja näyttävät hyviltä kaikilla laitteilla vaatii paljon työtä ja saattaa olla erittäin hankalaa. Vertailun vuoksi *Applen* iOS-käyttöjärjestelmä ei ole avoin ja iOS-laitteita valmistaa vain Apple itse, minkä takia fragmentaatio ei ole niin suuri ongelma iOS-sovelluskehitykselle.



Kuva 3.2: Android laitevalmistajien markkinaosuudet heinäkuussa 2013 [20].

3.1.1 Käyttöjärjestelmäfragmentaatio

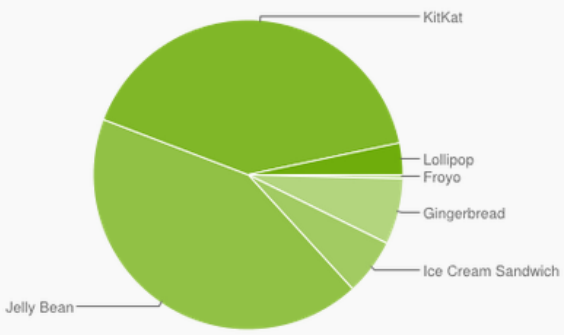
Androidin fragmentaatio käyttöjärjestelmätasolla tarkoittaa sitä, että eri käyttöjärjestelmäversioita on samalla hetkellä käytössä eri laitteissa. Android päivittyy tasaiseen tahtiin kuten taulukosta 2.1 voidaan nähdä ja käyttöjärjestelmäversioiden laitejakauma näyttää kuvan 3.3 mukaiselta.

Miksi kaikkia laitteita ei sitten päivitetä samantien uusimpaan versioon? Yksi syy on se, että tehottomimmat laitteet eivät yksinkertaisesti kykene suorittamaan uudempia Androidin versioita, joten niille ei tietoisesti julkaista uusia päivityksiä ja ne jäävät jumiin vanhaan versioon. Toinen syy on se, että eri laitevalmistajia on paljon, ja käyttöjärjestelmäpäivitysten jakaminen kuluttajalaitteisiin on jätetty heidän vastuulleen, koska he haluavat yleensä räätälöidä käyttöjärjestelmän itsensä näköiseksi ja lisätä siihen omia sovelluksiaan. Hyviä esimerkkejä tästä ovat Samsungin *TouchWiz*- ja HTC:n *Sense* -käyttöliittymät. Kun uusi Android-versio julkaistaan, saattaa kestää hyvin kauan ennen kuin laitevalmistajat saavat

oman versionsa jakoon omille laitteillensa. Usein uutta versiota ei päätetä julkaista vanhimmille laitemalleille, vaikka niissä tehoa riittäisikin, koska laitevalmistajat haluavat kuluttajien ostavan uudemman laitteen.

Androidin versiokehityksen myötä sen ohjelmointirajapinta muuttuu; rajapintoja vanhentuu ja uusia lisätään. Laitevalmistajat ovat voineet toteuttaa joitakin laiterajapintoja väärin, jolloin sama sovellus voi käyttäytyä eri tavalla eri laitteissa.

Version	Codename	API	Distribution
2.2	Froyo	8	0.4%
2.3.3 - 2.3.7	Gingerbread	10	6.9%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	5.9%
4.1.x	Jelly Bean	16	17.3%
4.2.x		17	19.4%
4.3		18	5.9%
4.4	KitKat	19	40.9%
5.0	Lollipop	21	3.3%



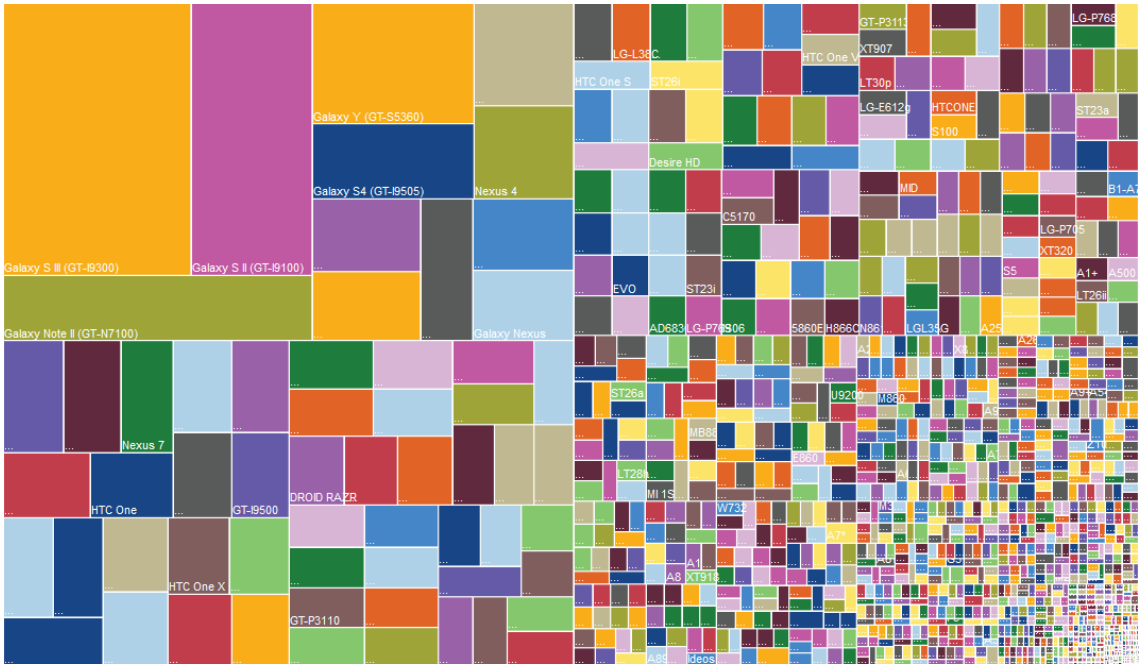
Kuva 3.3: Androidin käyttöjärjestelmäfragmentaatio 2.3.2015 [21].

3.1.2 Laitefragmentaatio

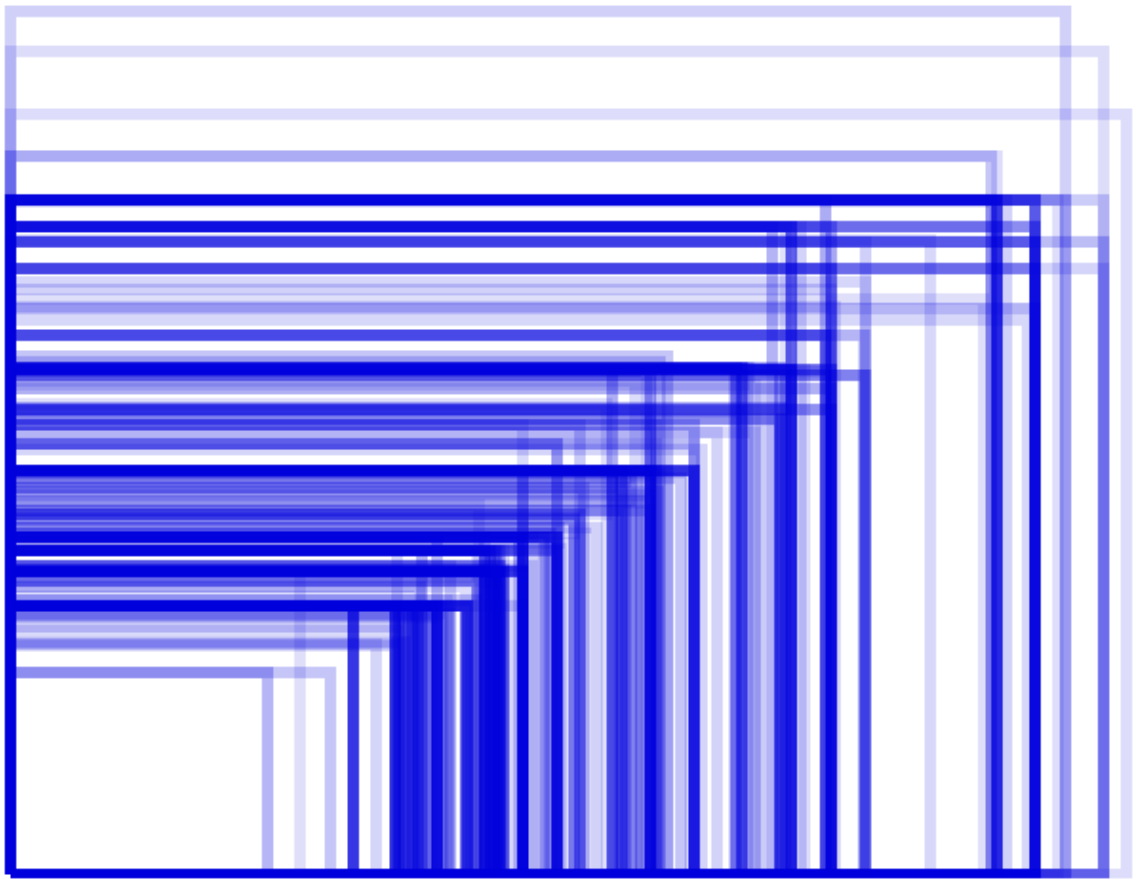
Laitetason fragmentaatiolla tarkoitetaan sitä, että Android-laitteita on paljon erilaisia, kuten kuvasta 3.4 voidaan nähdä. Erimallisia laitteita oli lähes 12000 vuonna 2013 [20]. Laitteissa on eri kokoisia näyttöjä, eri määrä muistia ja yleensäkin eri kombinaatio ominaisuuksia kuten kamera, anturit, muistikorttipaikka, puhelin, GPS ja radio. Eniten lisätyötä kehittäjille aiheuttaa kuvan 3.5 mukainen näyttöresoluutioiden kirjo. Android -sovellusten käyttöliittymätestaus eri kokoisilla laitteilla on paljon työläämpää kuin esimerkiksi *Applen* laitteilla, koska niissä eri näyttökokoja on vain kourallinen, kuten kuvasta 3.6 voidaan nähdä. *Applen* laitteista löytyy tämän diplomityön kirjoitushetkellä siis vain muutamaa eri kuvasuhdetta ja näyttökokoa, joten käyttöliittymäsuunnittelu ja -testaus on helpompaa.

3.2 Android-fragmentaation hyvä puolia

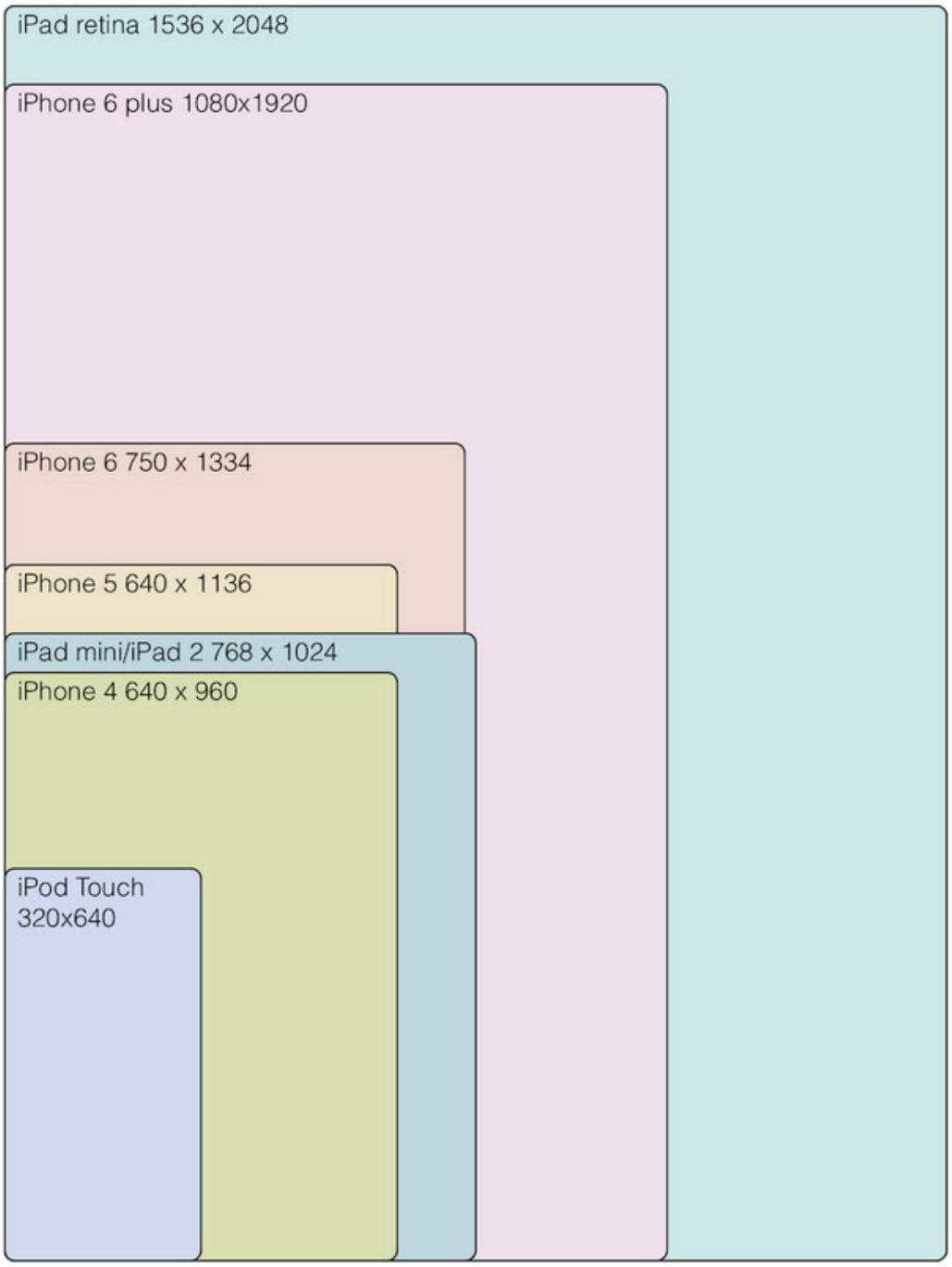
Androidin fragmentaatio ei ole pelkästään negatiivinen asia, vaan se tuo mukanaan myös monia hyötyjä sekä kehittäjille että kuluttajille. Halpojen Android-laitteiden olemassaolo on johtanut siihen, että kehittäjillä on entistä suurempi maailmanlaajuinen kohdeyleisö. Kaikkien laitteiden tukeminen on hankalaa,



Kuva 3.4: Android-laitteiden fragmentaatio kesäkuussa 2013 [20]



Kuva 3.5: Android-laitteiden näyttökoot [20]



Kuva 3.6: iOS-laitteiden näyttökoot [14]

mutta siihen panostaminen voi olla vaivan arvoista. Kuluttajat sen sijaan voivat hankkia hinnaltaan, kooltaan ja ominaisuuksiltaan juuri sopivan laitteen. [20]

3.3 Android-yhteensopivuusohjelma

Androidin on tarkoitus tarjota avoin kehitysalusta innovatiivisten sovellusten kehittämiseen. Laitteiden käyttäjät haluavat laajan valikoiman hyviä sovelluksia. Hyvät sovellukset tulevat sovelluskehittäjiltä, joita motivoi suuri Android-laitteita käyttävä kuluttajakunta. Laitevalmistajat sen sijaan luottavat siihen, että laaja sovellusvalikoima kasvattaa laitteiden suosiota. [22]

Androidin yhteensopivuusohjelma (engl. *Android compatibility program*) pyrkii määrittelemään alustan tekniset yksityiskohdat ja tarjoamaan laitevalmistajille työkaluja, joiden avulla sovelluskehittäjien sovellukset toimisivat mahdollisimman monella eri laitteella. Useat laitevalmistajat kehittävät itse sovelluksia laitteillensa, mutta ne eivät yksin riitä kuluttajille. Siksi on tärkeää, että sovelluskehittäjät pystyvät tuottamaan uusia sovelluksia, mikä edellyttää laitevalmistajilta yhteensopivia laitteita. Mitä yhteensopivamman laitteen laitevalmistaja julkaisee, sitä useampi sovellus siinä toimii, mikä taas on tärkeä edellytys laitteen kuluttajasuosiolle. Androidin yhteensopivuusohjelman tavoitteita ovat [22]:

- **Tarjota johdonmukainen sovellus- ja laiteympäristö sovelluskehittäjille.** Yhteensopivuusohjelman ansiosta sovelluskehittäjät voivat luottaa laite- ja käyttöjärjestelmärajapintojen toimivan dokumentaation kuvaamalla tavalla. Ilman johdonmukaisesti toimivia laitteita sovelluskehittäjät joutuisivat luomaan sovelluksistaan eri versioita eri laitteille.
- **Mahdollistaa kuluttajille johdonmukainen käyttökokemus.** Jos jokin sovellus toimii hyvin jollakin yhteensopivalla laitteella, tulee sen toimia yhtä hyvin myös millä tahansa muulla yhteensopivalla laitteella, jossa on sama version Androidista. Koska Android-laitteet eroavat toisistaan laite- ja sovellusominaisuuksiltaan, tarjoaa yhteensopivuusohjelma myös työkaluja sovellusten suodattamiseksi sovelluskaupoissa, jotta vain yhteensopivat sovellukset näytetään laitteille.
- **Antaa laitevalmistajille mahdollisuus erottua muista laitevalmistajista yhteensopivuudesta tinkimättä.** Androidin yhteensopivuusohjelma keskittyy sovelluskehittäjien sovellusten tukemista edesauttaviin seikkoihin, mikä antaa laitevalmistajille mahdollisuuden luoda ainutlaatuisia laitteita, jotka ovat silti yhteensopivia.
- **Minimoida yhteensopivuuden tukemiseen kuluvia resursseja.** Yhteensopivuuden takaamisen tulisi helppoa ja edullista laitevalmistajille. Yhteensopivuusohjelman testaustyökalu ovat ilmainen ja sen lähdekoodi on avoin.

Se on suunniteltu jatkuvaan testaukseen laitteen kehitysaikana. Lisäksi yhteensopivuusohjelma ei vaadi minkäänlaisia sertifikaatteja, joten sen käyttäminen on täysin ilmaista.

Android-yhteensopivuusohjelma koostuu kolmesta osasta [22]:

1. Androidin lähdekoodi.
2. Yhteensopivuuden määrittelydokumentti (engl. *Compatibility Definition Document*), CDD. Jokaiselle Androidin versiolle (ks. 2.1) on oma yhteensopivuuden linjaukset sisältävä CDD. CDD:n tarkoitus on koota ja selventää tietyt vaatimukset. Sen ei olekaan tarkoitus kattaa koko Android-lähdekoodia, vaan ainoastaan ne julkiset rajapinnat, joita esimerkiksi sovelluskehityksessä käytetään.[22]
3. Yhteensopivuuden testisarja (engl. *Compatibility Test Suite*, CTS. CTS on ilmainen testisarja Android-yhteensopivuuden testaamiseksi. CTS koostuu yksikkötesteistä, joita ajetaan työpöytäkoneella jatkuvasti laitekehityksen aikana. Sen tarkoitus on löytää mahdolliset yhteensopivuusongelmat mahdollisimman aikaisin, jotta laite pysyy yhteensopivana koko kehitysprosessin aikana [22]. CTS todentaja (engl. *CTS Verifier*) täydentää CTS:a. Se sisältää testejä, joita ei voida suorittaa paikallaan olevalle laitteelle. Tällaisia ovat esimerkiksi äänenlaatu- ja kiihtyvyysanturitestit, joille täytyy antaa testisyötettä käsin [22].

4. ONGELMIA JA RATKAISUJA

Kehittäjän on lähes mahdotonta omistaa kaikkia mahdollisia Android laitteita, saati sitten testata sovellustaan jokaisella niistä. Emulaattoreilla voi testata helposti sovelluksen logiikan ja miltä sovellus näyttää eri kokoisilla näytöillä, mutta loppujen lopuksi testaus tulisi tehdä oikeilla laitteilla. Yleensä kehittäjällä on muutama tai ehkä vain yksi laite, jolla testata sovellustaan. Niillä testaaminen ei takaa sovelluksen oikeaa toimintaa muilla laitteilla. Tässä luvussa käydään läpi Androidin fragmentaatiosta johtuvia ongelmia ja ratkaisuja kehittäjien näkökulmasta.

4.1 Käyttöjärjestelmätaso

Käyttöjärjestelmätason ongelmat johtuvat siitä, että eri Androidin versioissa on eroja. Tästä johtuen jotkin sovellusominaisuudet eivät toimi samalla tavalla kaikilla Androidin versioilla.

4.1.1 Android WebKit

Androidin WebKit on moduuli, joka tarjoaa Internetin selaamiseen liittyviä sovelluskehitystyökaluja. Hyvä esimerkki Androidin API-fragmentaatiosta on WebKit-moduulin **toDataURL**-funktion toteutus HTML5:n **canvas**-elementtiä vastaavalle **HTMLCanvasElement**-luokalle. HTMLCanvasElement-luokan dokumentaation mukaan funktion tulisi palauttaa elementin sisältö [**RFC 2397**]:n mukaisena URL-merkkijonona [23].

Androidissa funktion toteuttaa sisäisesti *ImageBufferAndroid* -luokka. Funktion toteutus on pysynyt tynkänä vielä Android 2.3.7 versioon asti kuten listauksesta 4.1 voidaan nähdä. Se siis palautti aina merkkijonon: "data:,", jollainen pitäisikin palauttaa tilanteissa, joissa canvas-elementillä ei ole yhtään pikseliä. Vasta Androidin versiossa 4.0 sille tuli jonkinlainen toteutus. Listauksessa 4.2 on funktion toteutus Androidin versiossa 4.3. Funktio ei tosin huomioi annettuja parametreja, vaan palauttaa kuvan aina PNG-formaatissa. Parametrien avulla funktiolle voitaisiin määrittää ulostuloksi JPEG-formaatti ja JPG-kuvan laatu.

Kehittäjä, joka kehittää esimerkiksi HTML-pohjaista hybridisovellusta, saattaa törmätä tähän ongelmaan. Sovellus toimii oikein laitteilla, joissa on Android 4.0 tai uudempi, mutta vanhemman Androidin omaavilla laitteilla canvas-elementin kuvan generointi ei toimikaan. Jos funktioon lisätään tulevaisuudessa JPEG-tuki, täytyy kehittäjän varautua siihen, että funktio ei toimi ollenkaan, tai se palauttaakin kuvan PNG-formaatissa.

Listaus 4.1: Android 2.3.7 WebKitin ImageBuffer-luokan toteutuksesta [24].

```
String ImageBuffer::toDataURL(const String&) const
{
    // leaving this unimplemented, until I understand what its for
    //(and what it really is).
    return "data:,"; // I think this means we couldn't make the data url
}
```

Listaus 4.2: Android 4.3 WebKitin ImageBuffer-luokan toteutuksesta [25].

```
String ImageBuffer::toDataURL(const String&, const double*) const
{
    // Encode the image into a vector.
    SkDynamicMemoryWStream pngStream;
    const SkBitmap& dst =
    imageBufferCanvas(this)->getDevice()->accessBitmap(true);
    SkImageEncoder::EncodeStream(&pngStream, dst,
                                 SkImageEncoder::kPNG_Type, 100);

    // Convert it into base64.
    Vector<char> pngEncodedData;
    SkData* streamData = pngStream.copyToData();
    pngEncodedData.append((char*)streamData->data(), streamData->size());
    streamData->unref();
    Vector<char> base64EncodedData;
    base64Encode(pngEncodedData, base64EncodedData);
    // Append with a \0 so that it's a valid string.
    base64EncodedData.append('\0');

    // And the resulting string.
    return String::format("\%s",
                          base64EncodedData.data());
}
```

Ratkaisuja tähän ongelmaan on kaksi. Helpompi ratkaisu on karsia tuki laitteilta, joissa on vanhempi Android-versio kuin 4.0. Tällä voidaan varmistaa, että sovellus ei ainakaan toimi väärin laitteilla, joille sen voi asentaa. Tämä onnistuu asettamalla manifestin (ks. 2.2) **uses-sdk**-elementin **minSdkVersion** -attribuutti arvoon 14, joka siis vastaa Androidin versiota 4.0.

Jos tuki vanhemmille Android-versioille halutaan kuitenkin pitää, voidaan **toDataURL**-funktio toteuttaa itse. Tähän löytyy valmiita toteutuksia, kuten **todataurl-png-js** -kirjasto [26].

4.1.2 Uudet sovelluskomponentit

Androidin kehittyessä siihen tulee uusia sovelluskomponentteja, kuten esimerkiksi käyttöliittymäkomponentteja. Uusien komponenttien käyttäminen edellyttää, että sovelluksen pienin vaadittava Android-versio on vähintään se, jossa käytettävä komponentti on julkaistu. Tämä johtaa siihen, ettei sovellusta voi asentaa laitteisiin, joissa on vanhempi Androidin versio.

Ratkaisu tähän onglmaan on käyttää Googlen tarjoamia *Support*-kirjastoja. Näiden kirjastojen eri versiot sisältävät uusimmista sovelluskomponenteista tärkeimmät, jotta niitä voisi käyttää vanhemmillakin Android-versioilla. Taulukossa 4.1 on esitelty eri support-kirjastot ja niiden sisältämät sovelluskomponentit. Kirjaston versionumero kertoo pienimmän API-version, jolla kirjasto toimii. Esimerkiksi **v4-support**-kirjasto toimii Androidin versiolla 1.6 (API 4) ja uudemmilla. Google suosittelee kirjastojen **V4-support**- ja **v7-appcompat**-kirjastojen käyttämistä koska ne tukevat monia Androidin versioita, ja ne tarjoavat rajapinnat suositeltujen käyttöliittymä ratkaisujen tuoteuttamiseen [27].

Taulukko 4.1: Support-kirjastot ja niiden sovelluskomponentit. [27]

Kirjasto	**Tuki sovelluskomponenteille**
v4-support	mm. Fragment, ViewPager, DrawerLayout, Loader ja FileProvider
v7-appcompat	ActionBar, ActionBarActivity, ShareActionProvider
v7-cardview	CardView
v7-gridlayout	GridLayout
v7-mediarouter	MediaRouter, MediaRouterProvider
v7-palette	Palette
v7-recyclerview	RecyclerView
v8-renderscript	RenderScript
v13-support	FragmentCompat
v17-leanback	BrowseFragment, DetailsFragment, PlaybackOverlayFragment, SearchFragment

Support-kirjastojen käyttäminen on helppoa. Esimerkiksi **Fragment**-luokan käyttäminen onnistuu linkittämällä **v4-support**-kirjasto projektiin, ja käyttämällä **Fragment**-luokkaa paketin *android.app.Fragment* sijasta paketista *android.support.v4.app.Fragment*.

4.1.3 Google Play Services

Kuten aiemmin mainittiin, Android-päivitysten jakelu on laitevalmistajien vastuulla. Tästä johtuen päivitysten saapuminen laitteille saattaa kestää jopa kuukausia. Pahimmassa tapauksessa päivitystä ei jaeta ollenkaan vanhoille laitemalleille. Tämä johtaa väistämättä Androidin fragmentoitumiseen käyttöjärjestelmätasolla.

Vuoden 2012 syyskuussa Google julkaisi Android-laitteille uudenlaisen jakelujärjestelmän nimeltä **Google Play Services**, jonka yksi päätarkoitus on torjua ohjelmistofragmentaatiota. Google Play Services tukee kaikkia Android-versioita aina versioon 2.2 asti ja se on järjestelmätason sovellus, jota ajetaan jatkuvasti taustalla. Lähes kaikki Googlen omat sovellukset vaativat Google Play Services -sovelluksen olemassaolon toimiakseen, koska ne käyttävät sen rajapintoja. Google Play Services tarjoaa myös sovelluskehittäjille rajapinnat googlen palveluihin. [28]

Miten Google Play Services pystyy torjumaan käyttöjärjestelmäfragmentaatiota? Koska se on järjestelmätason sovellus, sillä on oikeuksia tehdä asioita, joita tavalliset sovellukset eivät saa tehdä. Se voi esimerkiksi antaa itselleen lisää oikeuksia sekä päivittää itse itsensä ilman käyttäjän suostumusta. Tästä johtuen Google pystyy julkaisemaan uusia rajapintoja ja ominaisuuksia kaikille Google Play Services -sovellusta tukeville laitteille nopeasti ilman että päivitykset kiertävät laitevalmistajien kautta. Aiemmin Googlen täytyi paketoida uudet sovellukset käyttöjärjestelmäpäivitykseen uusien rajapintojen lisäksi, jos se halusi tuoda uusia ominaisuuksia Android-laitteille. Google Play Services -sovelluksen myötä riittää, että Google julkaisee uuden sovelluksen Google Play:ssä ja päivittää sen tarvitsemat uudet rajapinnat Google Play Services -sovellukseen. Ennen Googlen sovellukset olivat esiasennettuna käyttöjärjestelmään, mutta nykyään jokaisen niistä voi ladata ja päivittää erikseen Google Play:n kautta. Lisäksi Google on julkaissut ohjelmointirajapintoja omiin palveluihinsa ja sovelluksiinsa päivittämällä Google Play Services -sovellusta. [28]

Google Play Services vähentää tarvetta tuoda uusia käyttöjärjestelmäpäivityksiä, mutta ei tee niistä tarpeettomia. Uusia käyttöjärjestelmäpäivityksiä tarvitaan edelleen esimerkiksi laitteistotuen, Android-sovelluskehysrajapintojen sekä tiettyjen

turvallisuuskriittisten sovellusten päivittämiseksi tai lisäämiseksi [28]. Pitkällä aikavälillä Google Play Services tulee pienentämään käyttöjärjestelmäfragmentaatiota. Kun käyttöjärjestelmäpäivitysten tarve pienenee, niitä tulee harvemmin, mikä taas antaa aikaa kuluttajille päivittää laitteitaan uusimpia Android-versiota tukeviin laitteisiin.

4.1.4 Erikoisten Android-laitteiden tukeminen

Jotta Android sovelluksen saisi tarjolle mahdollisimman monelle käyttäjälle, ei pelkkä Google Play -julkaisu riitä. On olemassa paljon laitteita, joissa on räätälöity versio Android Open Source Projectista (AOSP). Näissä laitteissa ei ole tukea Googlen palveluille, joten niihin ei voi asentaa sovelluksia Google Play:sta. Jos tällaisia laitteita halutaan tukea, täytyy sovellus julkaista myös laitteen tukemassa sovelluskaupassa. Lisäksi, jos sovellus käyttää Googlen palveluja kuten karttoja, notifikaatioita tai sovelluksen sisäisiä maksuja, täytyy ne vaihtaa laitteen tukemiin vastaaviin ratkaisuihin. Alla on esimerkkejä tällaisista laitteista, ja mitä niiden tukemiseen vaaditaan.

Nokia X. Nokia julkisti helmikuussa 2014 kolme Android laitetta: Nokia X, Nokia X+ ja Nokia XL. Näissä laitteissa on käyttöjärjestelmänä Nokian räätälöimä versio Androidista. Jos oman sovelluksensa haluaa saataville näillä laitteilla, täytyy se julkaista Nokia Storessa. Google Play:ta varten tehdyn sovelluksen voi julkaista sellaisenaan Nokia Storessa, ellei se käytä Googlen karttoja, sovelluksen sisäisiä maksuja tai Googlen ilmoituspalvelua (push notifications). Tässä tapauksessa kyseiset ominaisuudet tarjoavat kirjastot täytyy korvata Nokian omilla vastaavilla kirjastoilla: *Nokia HERE* (kartat), *Nokia In-App Payment (IAP)* ja *Nokia Notifications*. Nokian kirjastoihin vaihtaminen on tehty helpoksi. Niiden rajapinnat vastaavat suurimmaksi osaksi Googlen vastaavia, mutta nimiavaruus on eri. Nokian mukaan tällaisen sovelluksen muokkaaminen Nokia Storeen sopivaksi vie keskimäärin alle 8 tuntia. [29]

Kindle Fire. Amazonin julkaisemat laitteet Kindle Fire, Kindle Fire HD ja Kindle Fire HDX ovat myös laitteita, jotka eivät tue Google Play:ta eikä Googlen palveluita, koska niissä on Android Open Source Projectista johdettu versio nimeltä *Fire OS*. Kindle Fire kehitystä varten Amazon tarjoaa oma SDK:n, jonka avulla sovellukseen saa kartat (*Amazon Maps API*), ilmoitukset (*Amazon Device Messaging API*) sekä sovelluksen sisäiset maksut (*Amazon In-App Purchasing API*).

4.2 Laitetaso

Laitetason fragmentaatio johtuu Android-laitteiden erilaisuudesta. Laitteissa on eri kokoisia näyttöjä, eri määrä muistia, eri kombinaatio oheislaitteita, ja eri valmistajien oheislaitteita. Sovelluksen toimiminen yhdellä tai kahdella Android-laitteella ei takaa siitä, että se toimisi samalla tavalla kaikilla Android-laitteilla.

4.2.1 Eri näyttökokojen tukeminen

Android-laitteita on monen kokoisia. Näyttöjen fyysinen koko sekä resoluutio vaihtelevat paljon, kuten kuvasta 3.5 (sivu 20) voidaan nähdä. Käyttäjälle on tärkeää, että sovellus näyttää siltä kuin se olisi suunniteltu juuri hänen laitteelleen. Kaikilla laitteilla testaaminen on kuitenkin mahdotonta, joten kehittäjän täytyy suunnitella sovellus niin, että se skaalautuu eri kokoisille näytöille mahdollisimman hyvin.

Androidin kehitysympäristö tarjoaa työkalut, joilla sovelluksen ulkoasun voi määrittää jokaiselle näyttökoolle sopivaksi. Sovelluksen käyttöliittymäelementit on mahdollista latoa eri tavalla erilaisille näytöille. Eri kokoisten näyttöjen tukemiseen kannattaa panostaa, jotta käyttäjälle tulisi tunne, että sovellus on suunniteltu juuri hänen laitteelleen. Android osaa skaalata käyttöliittymäkomponentteja ja muuttaa niiden kokoa, mutta jos näitä työkaluja ei käytä oikein, voi sovellus näyttää huonolaatuiselta tai se voi olla jopa rikki laitteilla, joilla sitä ei ole testattu.

Käyttöliittymäkomponenttien kokoja määriteltäessä tulisi käyttää fyysisten pikselien (*px*) sijasta **tiheysitsenäisiä pikseleitä** (engl. *density-independent pixel*). Yksi tiheysitsenäinen pikseli (*dp*) vastaa kooltaan yhtä fyysistä pikseliä näytöllä, jonka pikselitiheys (*dpi*) on **160**. Dp yksikköä käytettäessä Android skaalaa ajon aikana koot fyysisiksi pikseleiksi kaavalla

$$px = dp \cdot (dpi/160), \tag{4.1}$$

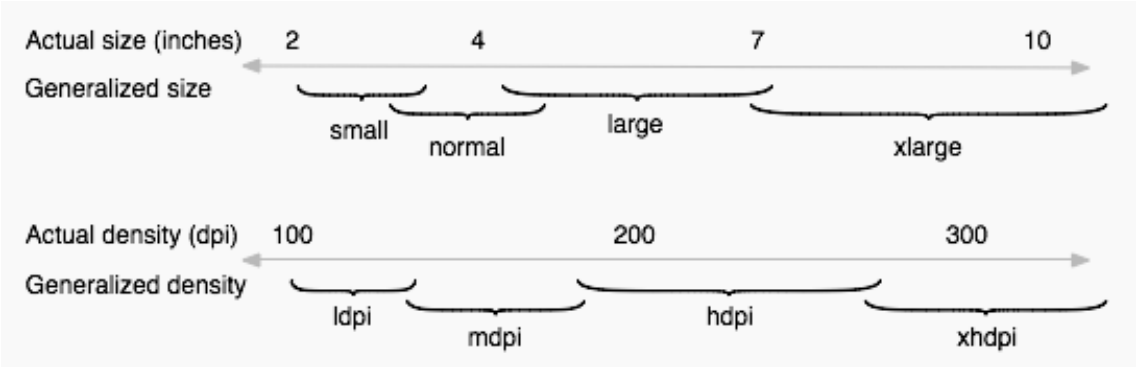
jolloin yksi dp vastaa esimerkiksi 320dpi näytöllä 2 fyysistä pikseliä. Lisäksi tekstien fonttikokoja määriteltäessä tulisi käyttää **skaalausitsenäisiä pikseleitä** (engl. *scale-independent pixel*). Skaalausitsenäiset pikselit (*sp*) järjestelmä skaalaa käyttäjän määrittämän tekstikoon mukaan. [30]

Jotta eri näyttökokojen tukeminen olisi helpompaa, Android yleistää laitteet neljään eri kokoluokkaan ja neljään eri tiheysluokkaan. Nämä luokat ovat:

- Näytön koko: *small*, *normal*, *large* ja *xlarge*.
- Näytön tiheys: *ldpi* (low), *mdpi* (medium), *hdpi* (high) ja *xhdpi* (extra high).

Androidin versiosta 3.2 (API 13) lähtien näyttökokoluokat on korvannut uusi tekniikka, jossa koot määritellään näytön leveyden mukaan.

Jokainen yleistetty näyttökoko- ja tiheysluokka käsittää tietyn alueen eri kokoja ja tiheyksiä. Esimerkiksi kahdella *normal*-kokoisella laitteella saattaa olla hieman eri kokoinen näyttö. Vastaavasti kahden *medium*-tiheyksisen laitteen pikselitiheys saattaa poiketa toisistaan jonkin verran. Näyttökoko- ja tiheysluokkien kattamat alueet näkyvät kuvassa 4.1.



Kuva 4.1: Näyttökoko ja -tiheysluokkien käsittämän karkeat alueet [30].

Android projektin *manifest*-tiedostossa voidaan luetella näyttökoot, joita sovellus tukee. Tuen puuttuminen estää sovelluksen lataamisen laitteelle. Tuki eri kokoisille näytöille määritellään listauksen 4.3 mukaan.

Listaus 4.3: Näyttökokojen tukeminen

```
<!-- arvot ovat oletuksena "true" -->
<supports-screens
     android:smallScreens=["true" | "false"]
     android:normalScreens=["true" | "false"]
     android:largeScreens=["true" | "false"]
     android:xlargeScreens=["true" | "false"]/>
```

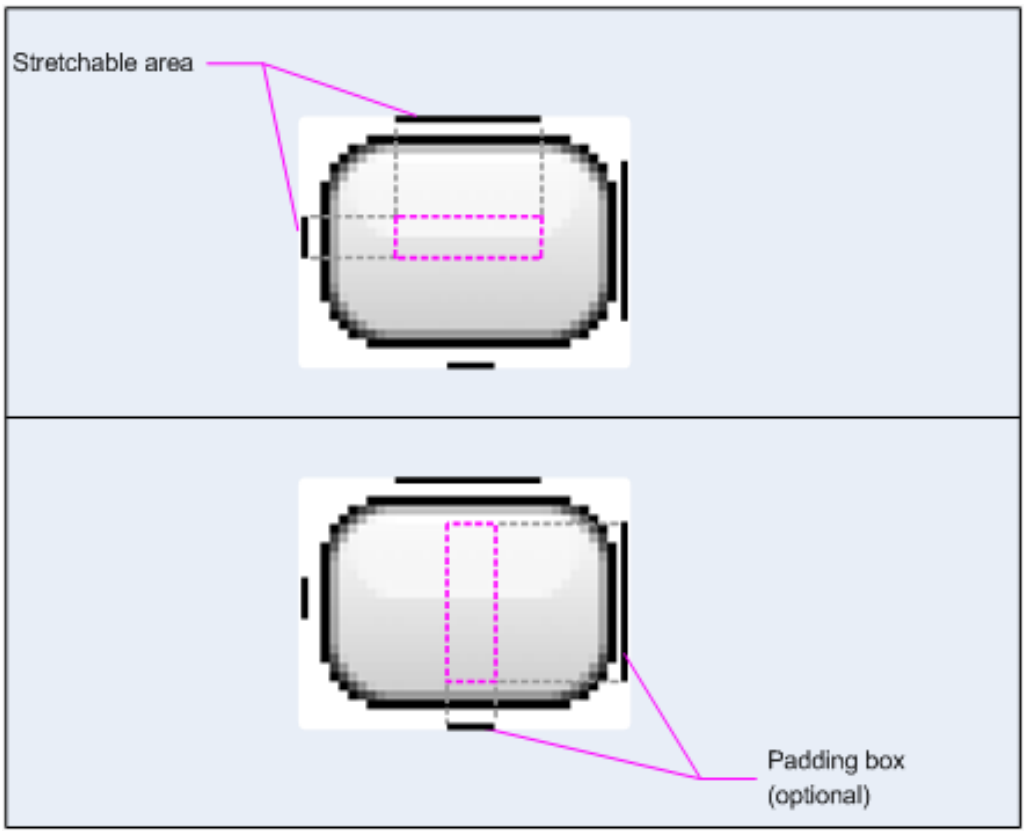
Jotta eri näyttökokoja ja -tiheyksiä voidaan tukea, tulee projektiin sisällyttää vaihtoehtoisia resursseja. Android järjestelmä valitsee ajon aikana sopivimman re-

surssiin riippuen näytön ominaisuuksista.

Eri kokoisille näytöille voi lisätä kokokohtaisia ladontatiedostoja. Oletuskansio, josta Android lataa ladontatiedostot on **res/layout**. Jos esimerkiksi *xlarge*-kokoisille näytöille halutaan oma ladontatiedosto, luodaan se kansioon **res/layout-xlarge**. Androidin versiosta 3.2 (API 13) lähtien edellä mainitut näyttökokoluokat ovat vanhentuneet ja uusi tekniikka jaottelee resursseja näytön leveyden mukaan. Jos sovellus tarjoaa esimerkiksi tableteille oman ladontatyylin, joka edellyttää näytöltä vähintään 600dp:n leveyttä vaaka-asennossa, luodaan kyseinen ladonta tiedosto kansioon **res/layout-sw600dp**. Määreen osa *sw* tulee sanoista *smallest width*. Tässä kansiossa olevia ladontatiedostoja käytetään, vaikka leveyttä olisikin alle 600dp laitteen ollessa pystyasennossa. Jos ladonnan tulee vaihtua vaaka- ja pystyasennon välillä, voidaan sama tiedosto laittaa sen sijaan kansioon **res/layout-w600dp**, jolloin tiedostoa käytetään vain kun näytöllä on asennosta riippuen tilaa leveyssuunnassa vähintään 600dp.

Projektiin tulee sisällyttää eri kokoiset kuvaresurssit eri näyttötiheyksille. Tämä koskee vain bittikarttatiedostoja. Oletuskansio kuvaresursseille on **res/drawable**. Oletuskansion kuvia käytetään skaalattuina niissä tilanteissa, joissa sopivampaa kuvaa ei löydy tiheysluokkakohtaisista kansioista. Jos kuvaresurssi puuttuu näytön edellyttämästä tiheysluokan kansiosta (esim. **res/drawable-hdpi**), Android järjestelmä etsii sopivampaa kuvaresurssia viereisistä tiheysluokan kansioista (**res/drawable-mdpi** ja **res/drawable-xhpi**), jos sellainen löytyy ja skaalaa sitä. Skaalaus saattaa aiheuttaa vääristymiä kuviin. Jotta kuvat näyttäisivät aina hyviltä, kannattaa jokaiseen tiheysluokan kansioon laittaa sopivan kokoinen kuvaresurssi. Tiheysluokkien kuvaresurssien koot tulisivat olla **3:4:6:8** suhteessa toisiinsa. Esimerkiksi jos *mdpi*-laiteella 100x100px kokoinen kuva on sopivan kokoinen, niin vastaava *ldpi*-kuva olisi 75x75px, *hdpi*-kuva olisi 150x150px ja *xhdpi*-kuva olisi 200x200px kokoinen.

Jos käyttöliittymässä on elementtejä, joiden koko vaihtelee paljon sovelluksen sisällä (esim. painikkeet), kannattaa niiden taustakuvat luoda **9-patch**-tiedostoksi. *9-patch*-tiedostot ovat **.9.png**-päätteellisiä. Kuvassa 4.2 on havainnollistettu *9-patch*-kuvan rakennetta. *9-patch*-kuva on png-kuva, jota skaalataan eri tavalla kuin tavallisia png-kuvia. *9-patch*-kuvalla on ylimääräinen yhden pikselin paksuinen läpinäkyvä tai valkoinen kehys. Kehyksen yläreunaan ja vasempaan reunaan piirretään mustalla värillä viivat, jotka kuvaavat alueita, joita venytetään tarvittaessa, kunnes kuva on skaalattu oikean kokoiseksi. Alareunaan ja oikeaan reunaan on mahdollista määrittää alue, jonka sisään näkymän sisältö tulee mahduttaa. [31]



Kuva 4.2: 9-Patch-kuva [31].

Kuvassa 4.3 on hyvä esimerkki 9-patch-kuvasta. Kuvassa on puhekupla, joka skaalautuu niin, että sen oikean alareunan nystyrä pysyy oikeassa alareunassa skaalauksesta huolimatta. Kuva 4.4 näyttää, miltä kuvan 4.3 9-patch-kuva näyttää eri kokoihin skaalattuna.

Kuva 4.3: Esimerkki: Puhekupla 9-patch-kuvana.

Kuva 4.4: Kuvan 4.3 9-patch-kuva Androidin skaalaamana.

4.2.2 Kuvan valitseminen

Varsin yleinen sosiaalista mediaa hyödyntävien mobiilisovellusten käyttötapaus on se, että käyttäjä voi jakaa kuvan sovellukseen. Kuvan valintaa varten on yleensä kaksi vaihtoehtoa; kamera ja galleria. Jos käyttäjä valitsee kameran, avataan yleensä ensisijainen kamerasovellus, jonka avulla käyttäjä voi ottaa kuvan sillä hetkellä. Jos käyttäjä sen sijaan valitsee galleriavaihtoehdon, avataan ensisijainen galleriasovellus, josta kuva valitaan.

Androidissa helpoin tapa tuoda kuva sovellukseen on käyttää olemassa olevia kamera- ja galleriasovelluksia, jotka käynnistetään **aikeilla** (2.7.1). Kamerasovelluksen käynnistäminen on havainnollistettu listauksessa 4.4. Aikeelle on määritettävä tiedosto, johon kamera tallentaa kuvan, ja josta kuva voi lukea myöhemmin.

Listaus 4.4: Kameran käynnistäminen kuvan ottamista varten.

```
protected void capture()
{
    // Luodaan aie, jonka tarkoitus on valita kuva kamerasta.
    Intent intent = new Intent();
    intent.setAction(MediaStore.ACTION_IMAGE_CAPTURE);

    ... // Tarkista, että kamera-sovellus.

    // Luodaan väliaikainen tiedosto, johon kuva tallennetaan.
    String dir = Environment.DIRECTORY_PICTURES;
    File path = Environment.getExternalStoragePublicDirectory(dir);
    mOutputFile = File.createTempFile("capture", ".jpg", path);

    // Asetetaan kuvalle ulostulotiedoston URI.
    intent.putExtra(MediaStore.EXTRA_OUTPUT, Uri.fromFile(mOutputFile));
```

```
    // Käynnistetään kamerasovellus.
    int requestCode = REQUEST_IMAGE_CAPTURE;
    startActivityForResult(intent, requestCode);
}
```

Galleriasovelluksen käynnistäminen on havainnollistettu listauksessa 4.5. Aikeelle määritetään tiedoston MIME-tyyppi, joka on tässä tapauksessa *image/** eli mikä tahansa kuva.

Listaus 4.5: Gallerian käynnistäminen kuvan valintaa varten.

```
protected void pickFromGallery()
{
    // Luodaan aie, jonka tarkoitus on valita kuva galleriasta.
    Intent intent = new Intent();
    intent.setAction(Intent.ACTION_PICK);

    ... // Tarkista, että galleria-sovellus.

    // Käyttäjä saa valita vain kuvia.
    pickFromGalleryIntent.setType("image/*");

    // Käynnistetään galleriasovellus.
    int requestCode = REQUEST_IMAGE_FROM_GALLERY;
    startActivityForResult(intent, requestCode);
}
```

Kun kamera- tai galleriasovellus on käynnistetty, voi käyttäjä ottaa tai valita kuvan tai peruuttaa valitsemisen. Joka tapauksessa päädytään lopulta takaisin alkuperäisen sovelluksen aktiviteettiin (2.7.2), josta kamera- tai galleriasovellus käynnistettiin. Tieto valitusta kuvasta palautuu aktiviteetin metodiin *onActivityResult*. Valitun kuvan hakeminen tässä metodissa on havainnollistettu listauksessa 4.6. Kamerakuvan hakeminen on helppoa, koska kuvan voi hakea aiemmin määritellystä tiedostosta. Galleriakuvan hakeminen on hieman hankalampaa, koska vastauksena saadaan vain kuvan osoittava URI, joka ei turvallisuussyistä kerro kuvatiedoston tiedostopolkua. Kuva joudutaan hakemaan tiedostokuvaajan avulla.

Listaus 4.6: Kuvan valinnan käsittely.

```
@Override
protected void onActivityResult(int requestCode,
                                int resultCode,
                                Intent data)
{
    // Käyttäjä keskeytti kuvan valinnan kamera- tai galleria-sovelluksesta.
   if(resultCode != RESULT_OK) {
     return;
   }

    Bitmap image = null;

   // Kuva valittu kamerasta.
   if (requestCode == REQUEST_IMAGE_CAPTURE) {
     // Dekoodataan kuva aiemmin määritetystä tiedostosta.
     image = BitmapFactory.decodeFile(mOutputFile);

       // Poistetaan välikainen tiedosto.
       mOutputFile.delete();
   }
   // Kuva valittu galleriasta.
   else if(requestCode == REQUEST_IMAGE_FROM_GALLERY) {
     // Haetaan galleriakuvan URI.
     Uri fileUri = data.getData();
     // Luodaan tiedosto deskriptori.
     ContentResolver cr = getContentResolver();
     ParcelFileDescriptor pfd = cr.openFileDescriptor(fileUri, "r");
     FileDescriptor fd = pfd.getFileDescriptor();
     // Dekoodataan kuva tiedosto deskriptorista.
     image = BitmapFactory.decodeFileDescriptor(fd);
   }

    ... // Kuvan skaalaus näkymään sopivaksi.

    // Asetetaan kuva näkymään.
    mImageView.setImageBitmap(image);
}
```

Listauksesta 4.6 on tilan säästämiseksi jätetty pois kuvan skaalaus näkymään sopivaksi. Liian suuren kuvan asettaminen näkymään epäonnistuu ja aiheuttaa virheen: *Bitmap too large to be uploaded into a texture*.

Jos edellä mainittuja koodeja käyttäisi sovelluksessa, niin useimmiten valittu kuva näkyisi oikein kuvanäkymässä (*mImageView*), mutta esimerkiksi Samsungin laitteilla kuva näkyisi aina kuin se olisi otettu vaakasuunnassa. Tämä johtuu sii-

tä, että eri laitevalmistajien kamerat tallentavat kuvan eri asennossa laitteeseen. Useimmat laitteet tallentavat kuvan juuri siinä asennossa kun kuva on otettu, mutta esimerkiksi Samsungin laitteet tallentavat kuvan kuin se olisi otettu vasemman laidan vaakasuunnassa (laite kallistettu vastapäivään pystysuunnasta) ja lisäävät kuvan EXIF-tietoihin tiedon oikeasta asennosta. Kuvan EXIF-tiedot on mahdollista hakea ja kääntää kuva niiden perusteella oikein päin. Tämä voi helposti jäädä kehittäjältä tekemättä, jos hänen testilaitteet sattuvat kaikki kääntämään kuvan valmiiksi oikeinpäin. Lisäksi Androidin dokumentaatiossa ei ole mainintaa tästä mahdollisuudesta, eikä Android CDD määrittele selvästi kuvan asentoa tallennettaessa. Android 4.4:n CDD:ssä sanotaan kameran asennosta seuraavasti:

> *"Both front- and end-facing cameras, if present, MUST be oriented so that the the long dimension of hte camera aligns with the screen's long dimension.* ***That is, when the device is held in the landscape orientation, cameras MUST capture images in the landscape orientation****. This applies regardless of the device's natural orientation; that is, it applies to landscape-primary devices as well as portrait-primary devices."* [32]

Lihavoitu kohta lainauksesta vihjaisi siihen, että kuvan asennon pitäisi vastata laitteen asentoa kuvaushetkellä. Toisaalta jos ollaan pikkutarkkoja, niin lause ei mainitse mitään pystyasennossa otetuista kuvista.

Ratkaisu tähän ongelmaan on aina hakea kuvan EXIF-tiedoista kuvan oikea asento ja kääntää kuva tarvittaessa. Listauksessa 4.7 on havainnollistettu, kuinka EXIF-asento haetaan kuvatiedostosta. Asennon hakeminen onnistuu helposti käyttämällä **ExifInteface**-luokkaa.

Listaus 4.7: Kuvan asennon hakeminen kuvatiedostosta.

```
protected int orientationFromImageFile(File imageFile)
{
    // Haetaan kuvatiedoston EXIF-tiedot.
    ExifInterface exif;
    try {
        exif = new ExifInterface(imageFile.getAbsolutePath());
    } catch (Exception exception) {
        return ExifInterface.ORIENTATION_UNDEFINED;
    }

    // Palautetaan kuvan asento.
    int defaultOrientation = ExifInterface.ORIENTATION_NORMAL;
    return exif.getAttributeInt(ExifInterface.TAG_ORIENTATION,
```

```
                                defaultOrientation);
    }
```

Kuvan asennon hakeminen kuva-URI:sta on jälleen hieman hankalampaa, koska URI:sta ei ole mahdollista saada tiedoston polkua, joka tarvittaisiin **ExifInterface**-luokkaa varten. Sen sijaan asento joudutaan lukemaan **MediaStore**-sisällöntarjoalta (2.7.4), josta asento löytyy astelukuna. Kuvan asennon hakeminen kuva-URI:n perusteella on havainnollistettu listauksessa 4.8.

Listaus 4.8: Kuvan asennon hakeminen kuvan URI:sta.

```
protected int orientationFromImageURI(Uri imageUri)
{
    // Luodaan kursori kuvan tietojen asentosarakkeeseen mediatietokannassa.
    ContentResolver cr = context.getContentResolver();
    String orientationColumn = MediaStore.Images.ImageColumns.ORIENTATION;
    String columns = new String[]{orientationColumn};
    Cursor cursor = cr.query(imageUri, columns, null, null, null);

    // Jos asentoa ei löytynyt, oletetaan oikean asento.
    if (cursor.getCount() != 1) {
        cursor.close();
        return ExifInterface.ORIENTATION_NORMAL;
    }

    // Luetaan kuvan asento (asteissa).
    cursor.moveToFirst();
    int orientationDegrees = cursor.getInt(0);
    cursor.close();

    // Palautetaan oikean EXIF-asento.
    if(orientationDegrees == 90) {
        return ExifInterface.ORIENTATION_ROTATE_90;
    }
    else if(orientationDegrees == 180) {
        return ExifInterface.ORIENTATION_ROTATE_180;
    }
    else if(orientationDegrees == 270) {
        return ExifInterface.ORIENTATION_ROTATE_270;
    }
    else {
        return ExifInterface.ORIENTATION_NORMAL;
    }
}
```

Kun kuvan asento on selvillä, täytyy kuva vielä kääntää oikein päin. Kääntäminen on havainnollistettu listauksessa 4.9. Kuva käännetään rotaatiomatriisin avulla.

Listaus 4.9: Kuvan kääntäminen EXIF-asennon perusteella.

```
protected Bitmap getOrientationCorrectedImage(Bitmap image, int orientation)
{
    // Luodaan rotaatiomatriisi kuvalle.
    Matrix matrix = new Matrix();

    switch (orientation) {
        case ExifInterface.ORIENTATION_ROTATE_90:
            matrix.postRotate(90);
            break;
        case ExifInterface.ORIENTATION_ROTATE_180:
            matrix.postRotate(180);
            break;
        case ExifInterface.ORIENTATION_ROTATE_270:
            matrix.postRotate(270);
            break;
        default:
            // Kuva on jo oikeinpäin.
            return bitmap;
    }

    // Luodaan ja palautetaan käännetty kuva.
    return Bitmap.createBitmap(bitmap, 0, 0,
                bitmap.getWidth(), bitmap.getHeight(),
                matrix, true);
}
```

4.2.3 Oheislaitteiden puuttuminen

Android-laitteissa on monia oheislaitteita, kuten kamera, bluetooth, WiFi, eri anturit, GPS, infrapuna, mikki, NFC ja puhelin. Fragmentaatiosta johtuen eri laitteissa on eri kombinaatio oheislaitteita. Kaikissa laitteissa ei esimerkiksi ole kameraa tai NFC:tä. Jos kehitettävä sovellus käyttää oheislaitteita, niin se toimii vain laitteilla, joissa on tarvittavat oheislaitteet. Sovellus sammuu, jos se yrittää käyttää oheislaitetta, jota laitteessa ei ole. Jotta sovelluksen sammuminen oheislaitteiden puuttumisen takia voidaan estää, on siihen kaksi vaihtoehtoa.

Jos sovelluksen idea ei perustu oheislaitteiden käyttöön, niin oheislaitteen olemassaolo voidaan tarkista suorituksen aikana ja oheislaitetta käyttävä ominaisuus piilottaa niillä laitteilla, joilla oheislaite puuttuu. Esimerkki tällaisesta sovelluksesta voisi olla pikaviestisovellus, jolla voi mm. lähettää laitteen kameralla otetun kuvan. Tällaisen sovelluksen tärkein ominaisuus on kuitenkin kommunikoida, eikä ottaa kuvia. Kuvan ottamisen estäminen ei siten tee sovelluksesta käyttökelvotonta.

Jos sovelluksen idea perustuu vahvasti yhteen tai useampaan oheislaitteeseen, niin paras tapa on estää koko sovelluksen asentaminen laitteille, joista ei löydy tarvittavia oheislaitteita. Esimerkki tällaisesta sovelluksesta voisi olla viivakoodeja kameran avulla lukeva sovellus. Sovelluksesta ei olisi mitään hyötyä laitteilla, joissa ei ole kameraa. Sovelluksen asentaminen laitteisiin, joista puuttuu tarvittavat oheislaitteet, voidaan estää määrittämällä sovelluksen Android-manifestiin tarvittavat oheislaitteet *uses-feature*-elementeillä. Tässä tapauksessa kamera määriteltäisiin pakolliseksi seuraavasti:

```
<uses-feature android:name="android.hardware.camera"
              android:required="true" />
```

4.2.4 Anturit

Android-laitteissa on monenlaisia antureita. Ne mittaavat mm. laitteen liikettä, asentoa ja ympäristönsä ominaisuuksia. Android tukee kolmea anturikategoriaa:

- **Liikeanturit**. Liikeanturit mittaavat kiihtyvyyttä ja pyörimisliikettä kolmessa ulottuvuudessa. Liikeantureita ovat esimerkiksi akselometri, painovoima-anturi ja gyroskooppi.
- **Ympäristöanturit**.Ympäristöanturit mittaavat ympäristöolosuhteita, kuten ilmanpainetta, valoisuutta ja kosteutta. Ympäristöantureita ovat mm. barometri, fotometri ja kosteusanturi.
- **Sijaintianturit**. Sijaintianturit mittaavat laitteen fyysistä sijaintia. Näitä antureita ovat esimerkiksi asentoanturi ja magnetometri.

Anturit voivat olla fyysisiä tai synteettisiä. Fyysiset anturit ovat oikeita komponentteja laitteen sisällä, ja ne tuottavat arvonsa mittaamalla fyysisiä ominaisuuk-

sia. Synteettiset anturit eivät ole fyysisiä komponentteja, vaan ne johtavat arvonsa yhdestä tai useammasta fyysisestä anturista. Synteettisiä antureja ovat esimerkiksi lineaarista kiihtyvyyttä mittaava anturi ja painovoima-anturi. [33]

Antureita käyttävän Android-sovelluksen toteutus niin, että se toimisi samalla tavalla kaikilla Android-laitteilla, voi olla hankalaa. Tämä johtuu siitä, että kaikissa laitteissa ei ole samanlaisia antureita, joissain laitteissa ei välttämättä ole ainuttakaan anturia, ja lisäksi synteettisien antureiden tuottamien tietojen prosessointialgoritmien toteutusvastuu on laitevalmistajilla. Antureiden erilaisuuden ongelmaa havainnollistaa hyvin kuva 4.5, jossa eri valmistajien Android-laitteissa on käynnissä *Steady Compass*-sovellus, jonka pitäisi näyttää, missä pohjoinen on. Kaikki laitteet näyttävät eri suuntia ja ylärivin keskimmäisen laitteen kompassin neula pyörii ympyrää.

Kuva 4.5: *Steady Compass* -niminen Android-sovellus eri Android-laitteilla. [34]

4.3 Testaus

Ennenkuin Android-sovelluksen julkaisee Google Play:ssa tai muussa sovelluskaupassa, niin sitä kannattaa testata muillakin kuin omilla laitteilla ja emulaattoreilla. Koska useiden laitteiden hankkiminen on kallista, kannattaa testaamiseen käyttää siihen tarkoitetuja palveluita. Tässä luvussa esitellään lyhyesti kaksi erilaista testauspalvelua: *AppKudo* ja *Samsung Remote Test Lab*. Lopuksi opastetaan hallitsemaan tuettuja laitteita Google Play -sovelluskaupassa.

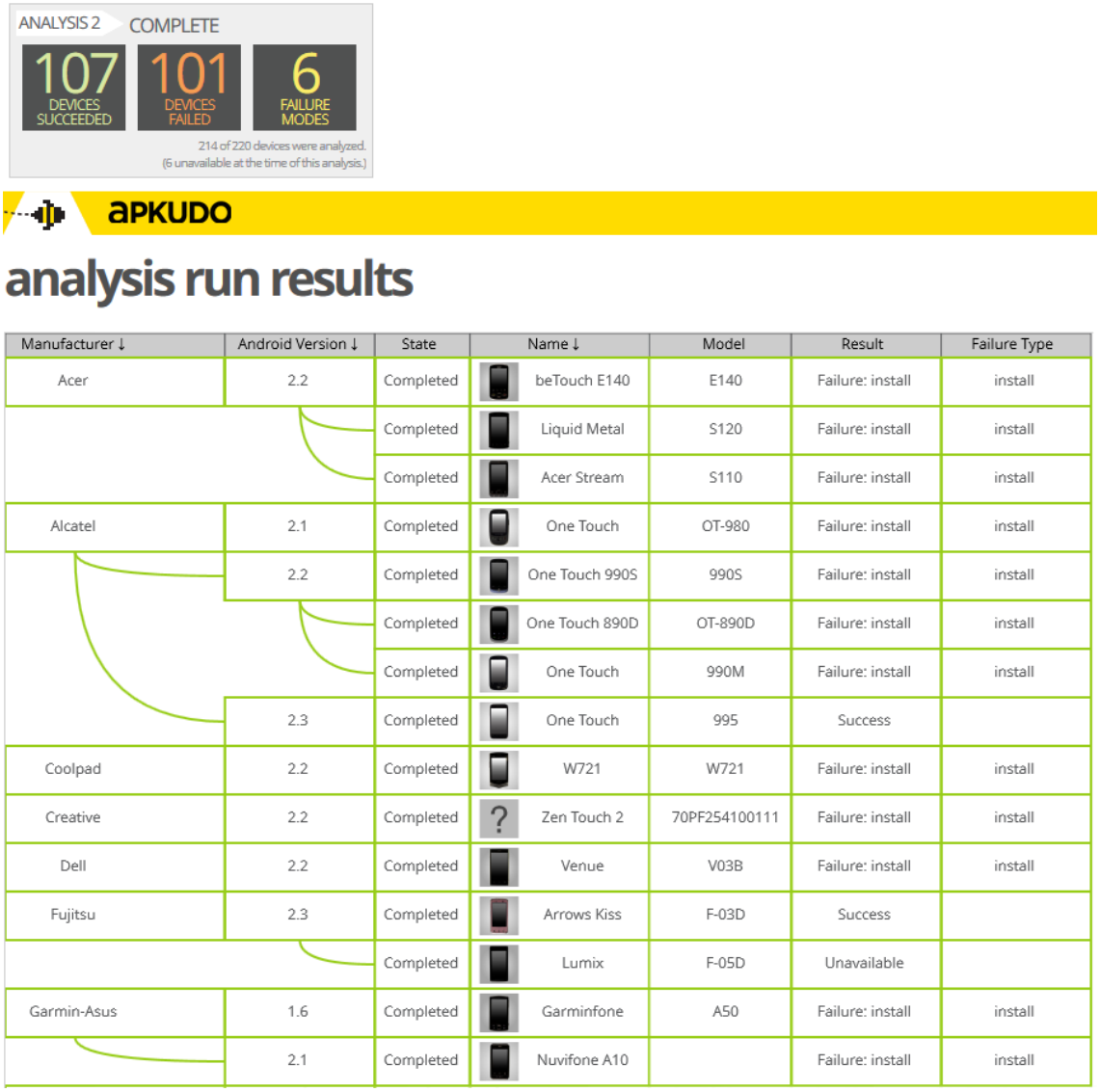
4.3.1 Appkudo

Appkudo on verkkopalvelu, johon kuka tahansa voi ladata Android-sovelluksen ja nähdä miten sovellus käyttäytyy palvelun laajalla laitevalikoimalla. Kun sovelluksen lataa palveluun, se asennetaan mahdollisen jonotuksen jälkeen kaikille palvelun fyysisille Android-laitteille. Jos asennus onnistui, annetaan sovellukselle satunnaisesti erilaisia syötteitä, kuten kosketus-, painallus- ja järjestelmätason tapahtumia käyttäen Googlen tarjoamaa *Monkey*-nimistä työkalua. Tämän jälkeen sovellus poistetaan laitteista. Lopuksi käyttäjälle näytetään testin tulokset (Kuva 4.6).

Tuloksia voi suodattaa mm. valmistajan, Android-version ja laitemallin mukaan. Kukin testitulos sisältää tiedon onnistumisesta, *Monkey*-työkalun tuottaman lokin, sovelluksen lokin, sekä mahdolliseen epäonnistumiseen johtaneen virheen tyypin ja tarkemmat tiedot. Appkudo asentaa ladatun sovelluksen kaikkiin laitteisiinsa, eikä siis ota huomioon sovelluksen manifestissa määriteltyjä rajoitteita. Tämän dokumentin kirjoitushetkellä Appkudo-palvelussa on yli 260 testilaitetta. [35]

4.3.2 Samsung Remote Test Lab

Samsung Remote Test Lab on verkkopalvelu, jossa käyttäjä voi varata Samsungin fyysisiä laitteita käyttöönsä. Kun laite on varattu, siihen voi ottaa etäyhteyden ja alkaa käyttää laitetta. Laitteeseen voi asentaa oman sovelluksensa raahaamalla *apk*-tiedoston etäyhteyssovellukseen. Etäyhteydestä johtuen laite vastaa syötteisiin kohtalaisella viiveellä ja ruudun päivitysnopeus on hidas, joten ajoitusta vaativien ominaisuuksien ja suorituskyvyn testaus ei onnistu.



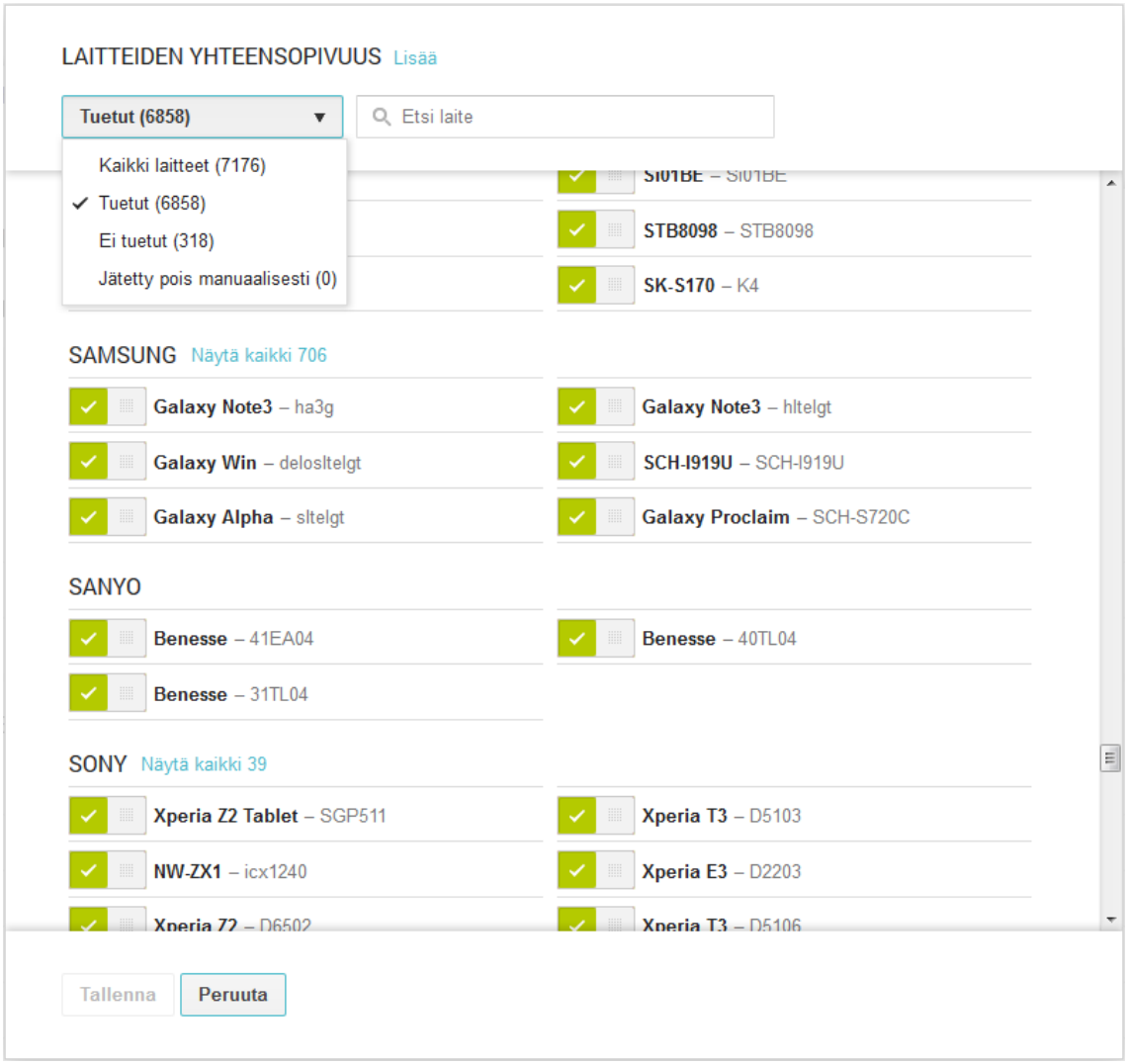
Manufacturer ↓	Android Version ↓	State	Name ↓	Model	Result	Failure Type
Acer	2.2	Completed	beTouch E140	E140	Failure: install	install
		Completed	Liquid Metal	S120	Failure: install	install
		Completed	Acer Stream	S110	Failure: install	install
Alcatel	2.1	Completed	One Touch	OT-980	Failure: install	install
	2.2	Completed	One Touch 990S	990S	Failure: install	install
		Completed	One Touch 890D	OT-890D	Failure: install	install
		Completed	One Touch	990M	Failure: install	install
	2.3	Completed	One Touch	995	Success	
Coolpad	2.2	Completed	W721	W721	Failure: install	install
Creative	2.2	Completed	Zen Touch 2	70PF254100111	Failure: install	install
Dell	2.2	Completed	Venue	V03B	Failure: install	install
Fujitsu	2.3	Completed	Arrows Kiss	F-03D	Success	
		Completed	Lumix	F-05D	Unavailable	
Garmin-Asus	1.6	Completed	Garminfone	A50	Failure: install	install
	2.1	Completed	Nuvifone A10		Failure: install	install

Kuva 4.6: Appkudo-testin tulokset. [35]

4.3.3 Tuettujen laitteiden hallinta Google Play:ssa

Kun sovelluksen lataa *Google Play*:hin, niin kehittäjäkonsolista voi nähdä sovelluksen kanssa yhteensopivat laitteet, eli laitteet, joille sovellus näytetään sovelluskaupassa. Kehittäjäkonsolista näkee myös laitteet, joille sovellusta ei ole mahdollista asentaa jostain syystä. Kuvassa 4.7 on esitetty kehittäjäkonsolin tuettujen laitteiden hallintapaneeli.

Hallintapaneelin avulla voi estää sovelluksen näkymisen sovelluskaupassa halutuille laitteille. Tämän on tarpeen silloin, kun Android-manifesti ei rajaa laitetta pois tuettujen joukosta, mutta sovellus ei jostain syystä toimi tietyillä laitteilla tarpeeksi hyvin. Näin voidaan estää huono käyttäjäkokemus näillä laitteilla.



Kuva 4.7: Tuettujen laitteiden hallinta Google Play:ssa.

5. YHTEENVETO

Tässä diplomityössä selvitettiin Androidin fragmentaatiota ja siitä johtuvia ongelmia, jotka tulisi ottaa huomioon sovelluskehityksessä. Työ aloitettiin Androidin esittelyllä sovelluskehittäjän näkökulmasta. Seuraavaksi selitettiin fragmentaatio terminä ja miten se liittyy Androidiin. Lopuksi käytiin läpi yleisimpiä fragmentaatiosta johtuvia ongelmia ja pyrittiin löytämään ratkaisuja niiden torjumiseksi.

Android on Googlen kehittämä avoimeen lähdekoodiin perustuva käyttöjärjestelmä, joka on suunnattu kosketusnäytöllisille mobiililaitteille. Androidin ensimmäinen versio julkaistiin syyskuussa vuonna 2008. Sen jälkeen Android on kehittynyt nopeasti. Vuonna 2013 eri laitemalleja oli lähes 12000. Tämän työn kirjoitushetkellä uusin versio on 5.0, joka julkaistiin lokakuussa vuonna 2014.

Fragmentaatio tarkoittaa pirstaloitumista, tai särkymistä pieniin osiin. Androidin sanotaan olevan fragmentoitunut, koska yhtä aikaa käytössä on paljon kooltaan, laitteistoltaan ja käyttöjärjestelmäversioltaan eroavia Android-laitteita. Framentaation vuoksi Android-sovellusten kehittäminen on hankalampaa, koska tuettavia laitteita ja Android-versioita on niin paljon. On lähes mahdotonta taata, että sovellus toimii yhtä hyvin kaikilla Android-laitteilla.

Pääsyy Androidin fragmentaatioon on se, että Android on avoimen lähdekoodin käyttöjärjestelmä. Kuka tahansa voi alkaa valmistamaan mobiililaitteita ja halutessaan räätälöidä oman versionsa Androidista niiden käyttöjärjestelmäksi. Se miksi kaikissa laitteissa ei ole uusinta tai edes toisiksi uusinta Androidin versiota, johtuu siitä, että päivitysten jakelu on laitevalmistajien vastuulla. Monet laitevalmistajat räätälöivät laitteisiinsa oman versionsa Androidista. Voi kestää jopa useita kuukausia, ennen kuin Googlen julkaisema Android-päivitys saapuu laitteille, joissa on räätälöity Android. Laitevalmistajat saattavat myös jättää vanhat laitteensa päivityksen ulkopuolelle taloudellisista syistä, tai jos uusi versio ei toimisi tarpeeksi hyvin vanhalla laitteella.

Google tarjoaa kehittäjille ja laitevalmistajille hyviä työkaluja, joilla fragmentaation aiheuttamia ongelmia voidaan ehkäistä. Esimerkiksi Androidin yhteen-

sopivuusohjelma auttaa laitevalmistajia toteuttamaan laiteensa niin, että sovelluskehitys ja käyttökokemus olisivat mahdollisimman johdonmukaisia. *Support*-kirjastojen avulla sovelluskehittäjän voivat sen sijaan tuoda uusimmat sovelluskomponentit vanhemmille laitteille.

Työssä käytiin läpi vain osa fragmentaation aiheuttamista ongelmista. Työtä voisi jatkaa lisäämällä uusia ongelmia ja ratkaisuja, ja päivittämällä olemassa olevia, jos ne vanhentuvat. Kaikki ongelmia on kuitenkin lähes mahdoton tietää, ennen kuin ne kohtaa. Jokainen uusi laite ja Android-versio voi tuoda mukanaan uusia ongelmia.

LÄHTEET

[1] Android-app-market.com. Android Architecture - The Key Concepts of Android OS. 17.2.2012. Android-app-market.com. [WWW]. [Viitattu 2.6.2013]. Saatavissa: http://www.android-app-market.com/android-architecture.html.

[2] Lucy Black. Android - An Illustrated History. 22.4.2013. i-programmer.info. [WWW]. [Viitattu 22.5.2013]. Saatavissa: http://www.i-programmer.info/news/193-android/5753-android-history.html.

[3] Gareth Beavis. A complete history of Android. 23.4.2008. Techradar.com. [WWW]. [Viitattu 22.5.2013]. Saatavissa: http://www.techradar.com/news/phone-and-communications/mobile-phones/a-complete-history-of-android-470327.

[4] developer.android.com. What is API Level. [WWW]. [Viitattu 28.9.2013]. Saatavissa: http://developer.android.com/guide/topics/manifest/uses-sdk-element.html.

[5] source.android.com. Licenses. [WWW]. [Viitattu 28.9.2013]. Saatavissa: http://source.android.com/source/licenses.html.

[6] developer.android.com. Dashboards - Platform versions. [WWW]. [Viitattu 8.6.2013]. Saatavissa: http://developer.android.com/about/dashboards/index.html.

[7] http://www.talkandroid.com. A Guide To The Android Market. [WWW]. [Viitattu 28.9.2013]. Saatavissa: http://www.talkandroid.com/guides/beginner/android-market-explained/.

[8] http://marketingland.com. Android Market Becomes Google Play, Reflects Google's Multiplatform Content Aims. [WWW]. [Viitattu 28.9.2013]. Saatavissa: http://marketingland.com/android-market-to-become-google-play-reflects-googles-multiplatform-content-aims-7297.

[9] https://support.google.com. Transaction fees. [WWW]. [Viitattu 27.4.2015]. Saatavissa: https://support.google.com/googleplay/android-developer/answer/112622.

[10] https://support.google.com. Return paid apps & games. [WWW]. [Viitattu 27.4.2015]. Saatavissa: https://support.google.com/googleplay/answer/134336.

[11] Ed Burnette. ZDnet.com. Java vs. Android APIs. [WWW]. [Viitattu 17.6.2013]. Saatavissa: http://www.zdnet.com/blog/burnette/java-vs-android-apis/504.

[12] http://developer.android.com. Get the Android SDK. [WWW]. [Viitattu 13.9.2014]. Saatavissa: http://developer.android.com/sdk/index.html.

[13] http://www.jetbrains.com. The Professional Android IDE. [WWW]. [Viitattu 13.9.2014]. Saatavissa: http://www.jetbrains.com/idea/features/android.html.

[14] http://carst.me/. Current iOS screen resolutions. [WWW]. [Viitattu 27.4.2015]. Saatavissa: http://carst.me/2014/09/current-ios-screen-resolutions/.

[15] developer.android.com. Application Fundamentals - Application Components. [WWW]. [Viitattu 8.6.2013]. Saatavissa: http://developer.android.com/guide/components/fundamentals.html.

[16] developer.android.com. Managing projects. [WWW]. [Viitattu 28.3.2015]. Saatavissa: http://developer.android.com/tools/projects/index.html.

[17] http://developer.android.com. App Manifest. [WWW]. [Viitattu 13.9.2014]. Saatavissa: http://developer.android.com/guide/topics/manifest/manifest-intro.html.

[18] http://developer.android.com. User Interface. [WWW]. [Viitattu 7.3.2014]. Saatavissa: http://developer.android.com/guide/topics/ui/.

[19] Kim, T., Adeli, H. et al. 2011. Software Engineering, Business Continuity, and Education. ISBN: 978-3-642-27206-6. 736 s.

[20] Open Signal. Android Fragmentation Visualized. Heinäkuu 2013. Opensignal.com. [WWW]. [Viitattu 18.8.2013]. Saatavissa: http://opensignal.com/reports/fragmentation-2013/.

[21] developer.android.com. Dashboards. [WWW]. [Viitattu 18.8.2013]. Saatavissa: http://developer.android.com/about/dashboards/index.html.

[22] http://source.android.com. Android Compatibility. [WWW]. [Viitattu 5.5.2014]. Saatavissa: http://source.android.com/compatibility.

[23] http://www.whatwg.org. The canvas element. [WWW]. [Viitattu 14.4.2014]. Saatavissa: http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html#dom-canvas-todataurl.

[24] https://android.googlesource.com. ImageBufferAndroid, Android 2.3.7. [WWW]. [Viitattu 14.4.2014]. Saatavissa: https://android.googlesource.com/platform/external/webkit/+/android-2.3.7_r1/WebCore/platform/graphics/android/ImageBufferAndroid.cpp.

[25] https://android.googlesource.com. ImageBufferAndroid, Android 4.3. [WWW]. [Viitattu 14.4.2014]. Saatavissa: https://android.googlesource.com/platform/external/webkit/+/android-4.3_r3.1/Source/WebCore/platform/graphics/android/ImageBufferAndroid.cpp.

[26] http://code.google.com. todataurl-png-js. [WWW]. [Viitattu 14.4.2014]. Saatavissa: http://code.google.com/p/todataurl-png-js/.

[27] developer.android.com. Support Library Features. [WWW]. [Viitattu 28.3.2015]. Saatavissa: http://developer.android.com/tools/support-library/features.html.

[28] http://arstechnica.com. Balky carriers and slow OEMs step aside: Google is defragging Android. [WWW]. [Viitattu 28.9.2013]. Saatavissa: http://arstechnica.com/gadgets/2013/09/balky-carriers-and-slow-oems-step-aside-google-is-defragging-android/.

[29] http://developer.nokia.com. Nokia X Platform overview. [WWW]. [Viitattu 8.8.2014]. Saatavissa: http://developer.nokia.com/nokia-x/platform-overview.

[30] http://developer.android.com. Supporting Multiple Screens. [WWW]. [Viitattu 29.9.2013]. Saatavissa: http://developer.android.com/guide/practices/screens_support.html/.

[31] http://developer.android.com. Canvas and Drawables. [WWW]. [Viitattu 3.11.2013]. Saatavissa: http://developer.android.com/guide/topics/graphics/2d-graphics.html.

[32] static.googleusercontent.com. Android 4.4 CDD. [PDF]. [Viitattu 2.6.2014]. Saatavissa: http://static.googleusercontent.com/media/source.android.com/fi//compatibility/4.4/android-4.4-cdd.pdf.

[33] http://developer.android.com. Sensors Overview. [WWW]. [Viitattu 13.9.2014]. Saatavissa: http://developer.android.com/guide/topics/sensors/sensors_overview.html.

[34] http://gameovenstudios.com. Bounden on Android delayed. [WWW]. [Viitattu 20.9.2014]. Saatavissa: http://gameovenstudios.com/bounden-on-android-delayed/.

[35] https://www.apkudo.com. Appkudo for developers. [WWW]. [Viitattu 20.9.2014]. Saatavissa: https://www.apkudo.com/developers.html.